BUCS (BADUSB Challenge System)

Executive Summary

Users need an easy way to input data to a computer. This led to the creation of devices such as mice and keyboards to allow users to achieve that goal. Throughout the years, technology of mice and keyboards alike grew, and we now have created USB and HID to assist in creating easy input methods for users. The issue with HID is that it is an innately trusted device for operating systems. While there is no issue with innately trusting a keyboard, there is a problem with trusting a device that can masquerade as a keyboard. In 2014 the power of devices known generically as BadUSBs was demonstrated at Defcon. These devices can pretend to be a keyboard and automatically input malicious commands upon plug in. The user is powerless to stop the USB after they plug the device into their machine since the takeover occurs in an incredibly short amount of time. This project seeks to create a small amount of required human interaction in the device handshake to mitigate the threat from a BadUSB.

This project can potentially save companies from headaches in the future. There is no current ethical issue with developing an application to assist in the protection of a computer, and while there are currently mitigations in place to protect against BadUSB attacks, this will introduce a full protection against it.

Project Goals

1. Determine optimal programming language
2. Build prototype of BUCS
3. Test prototype of BUCS

These goals are the overall goals of the project. The specifics of how they will be accomplished are to be discussed later in this plan, but our project can be expressed in these four main goals.

Project Methodology

Develop BUCS (BadUSB Challenge System) Prototype

- Goal 1: Determine programming language required for optimal program functionality
    - Sub Objective 1: Determine if Python optimal

    - Sub Objective 2: Determine if C optimal

- Goal 2: Program the detection of USB keyboard devices.
    - Sub objective 1: We will begin by researching ways to interact with device manager and enumerate devices connected to the PC.
- Goal 3: Program the ability for the program to lockout input from a peripheral device
    - Sub objective 1: We still begin researching the optimal way to lock-out a USB device for inputting information into the main system outside of the check without authorization.
- Goal 4: Program the ability for the program to unlock input from a peripheral device
    - Sub objective 1: We still begin researching the best way to undo the lock-out of a USB device for inputting information into the main system.

Test BUCS Prototype

- Goal 1: Test with just detection
    - Sub Objective 1: Test the effectiveness of plugging in a new keyboard while running BUCS with the expected outcome of succesful detection multiple times with multiple devices to ensure good coverage
    - Sub Objective 2: Test the effectiveness of plugging in a bad USB device while running BUCS with the expected outcome of succesful detection multiple times to ensure good coverage
- Goal 2: Test with detection and lock out
    - Sub Objective 1: Test the effectiveness of plugging in a new keyboard while running BUCS with the expected outcome of lock out multiple times with multiple devices to ensure good coverage
    - Sub Objective 2: Test the effectiveness of plugging in a bad USB device while running BUCS with the expected outcome of lock out multiple times to ensure good coverage
- Goal 3: Test with detection, lock out, and CAPTCHA style challenge
    - Sub Objective 1: Test the effectiveness of plugging in a new keyboard while running BUCS with the expected outcome of succesful detection, lock out, and deployment of CAPTCHA style challenge multiple times with multiple devices to ensure good coverage

    - Sub Objective 2: Test the effectiveness of plugging in a bad USB device while running BUCS with the expected outcome of succesful detection, lock out, and deployment of CAPTCHA style challenge multiple times to ensure good coverage

- Tentative Goal 4: If found fully effective, continue development and testing to a final product (detailed later in Package deliverable software for installation)

If found ineffective research what other options there are to develop a product for Windows

- Goal 1: Investigate potential Windows configurations that will halt USB communication until user selects otherwise.
- Goal 2: Consider available security policy objects to allow for mass deployment over a domain.

Package deliverable software for installation

- Goal 1: Package project into a zip folder for more general distribution.
    - Sub Objective 1: Prepare documentation for download and setup.
- Goal 2: Package project into files usable by applicable installation managers (apt, rpm, yum, msi).
    - Sub Goal 1: Achieve functionality on multiple platforms.

Results / Findings

- milestone 1 outcomes
    - Refined scope of problem
    - Refined our methodology BAD-USB/Documentation/Milestone 1/Proposal.md
    - Determined our inital development plan BAD-USB/Documentation/Milestone 1/Proposal.md
    - Determined project timeline BAD-USB/Documentation/Milestone 1/Proposal.md
    - Determined risks for project BAD-USB/Documentation/Milestone 1/Proposal.md
- milestone 2 outcomes
    - Began work on Windows programs as intro prototypes to the final product - BAD-USB/BUCS Dev start/Windows/Obsolete & BAD-USB/BUCS Dev start/Linux
    - Obtained event based detection in Windows - BAD-USB/BUCS Dev start/Windows /hid_test.xml

    - Crafted a number of BadUSB attacks to leverage against our BUCS - BAD-USB/BadUSB scripts/
    - Created a script that applies out intended defense mechanism - BAD-USB/BUCS Dev start/Windows/myscript.bat & /Cap.py
    - Began work on GUI - BAD-USB/BUCS Dev start/Windows/BUCS_GUI_FRAME.py
- milestone 3 outcomes
    - Created a new library to create the CAPTCHA that better fit needs of users BAD-USB/BUCS Dev start/Windows/dictionary.py && Cap.py
    - Researched BadUSB scripts more in depth and discovered common methods among them
    - Researched common BadUSB methods to better defend against them
    - Crafted ideas as to how additional protections could be implemented

Install Instructions

Requirements

1. python3
2. Windows 10 or newer

Installation Instructions

Step 1:

Download and install the appropriate python3 for your windows machine from `python.org/downloads/windows`

Step 2:

Enable PNP auditing in the Windows Group Policy Editor under:

```
Computer Configuration \ Windows Settings \ Security Settings \ Advanced Audit Policy Configur
```

Alt text
Enable Success and Failure.

Step 3: Download the 1.0_Pack.zip

https://github.com/abladow/BAD-USB/raw/main/Install/1.0_Pack.zip

Unzip the contents

You are ready to make changes to the code without throwing dependency errors!

Step 4: Configure Task scheduler

1. In task scheduler import the BUCS_Task.xml file:

```
Actions -> Import Task
```

2. Navigate to where BUCS_Task.xml is downloaded and select it
3. Set the group or user for it to run with if you wish to change it from default (Administrators) by selecting Change User of Group
4. Select the Actions tab at the top to configure the actions
5. The default action is a place holder to open a command prompt. Select the action and choose edit to change it.
6. Under Program/Script browse to where the BUCS download is and choose the BUCS_Script.bat file
7. Under triggers is where the the delay can be edited, this is important to have proper interjection to the attack, default is 5 seconds. This may need to change for your system. To do this navigate to the triggers tab and select the trigger. Then select edit. There you can edit the delay.
8. Hit Ok on the task window and close task scheduler

Step 5: Configure BUCS_Script.bat

1. Open the BUCS_Script.bat in a text editor of your choice.
2. On line 3 set the calue of the statements within the <> brackets
    i. The first <> statement will be the PATH to the python3 executable on your system
    ii. The second <> statement will be the PATH to the Cap.py file
3. Save and close

Getting started

If properly configured, when a HID device is plugged in to the system BUCS will activate