**Keystroke Injection Detection and Prevention**

Emory Lindsey

Georgia College & State University

Faculty Advisor: Dr. Gongbing Hong

Department of Computer Science and Information Systems

Keystroke injection is a hardware attack vector by which a device, usually in the form of a USB thumb drive, is used to type a sequence of keystrokes at relatively fast speeds. During device enumeration, such devices are typically enumerated as HID keyboard devices. Due to the absence of a standard for device authentication, the USB protocol causes a USB bus to blindly trust any connected device regarding its advertised capabilities [1]. Therefore, an injection device can easily trick the host machine into believing that it is simply a USB keyboard. The purpose of this research is to design a simple detection mechanism for keystroke injection attacks using Linux as the target platform. There are many ways to approach this problem and such approaches differ in terms of complexity. The approach taken here is relatively simple; however, as mentioned later, the detection model discussed below was found to be successful in a practical manner.

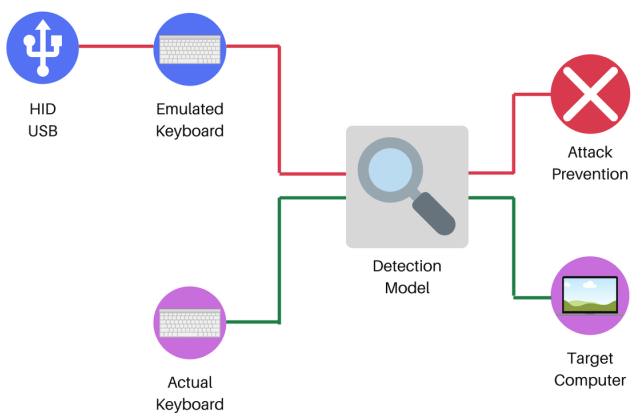


Figure 1: Visual Representation of Detection/Prevention Model

As depicted above in Figure 1, the detection model monitors all connected devices that are classified as keyboard devices. As shown above, any normal activity is simply passed to the target computer and any activity that is flagged as malicious will trigger the attack prevention mechanism, which will disconnect all present keyboard devices. In regards to implementation, a simple kernel module handles the process of capturing keystroke events, calculating the keystroke separation timing, denoted as *s*,between two subsequent keystrokes, and sending the timing data to user space using the netlink socket API as mentioned by Neuner [1]. The userland application is then responsible for the detection and prevention of injection attacks. Since human beings cannot obtain a value of *s* less than 80 ms [2], the detection threshold for the models is *s* = 80 ms. In an effort to reduce false positives, the userland application calculates the average separation timings, *savg*, across every four keystroke events. As mentioned before, computers often trust the keyboard input that is received. However, despite this seemingly large vulnerability, successful execution requires some degree of social engineering. Due to this requirement, keystroke injection devices are often programmed to perform injection at inhuman speeds in an effort to reduce the amount of time needed to perform the attack. These two truths lessen the burden of detecting and stopping this form of attack. Of course, in theory, an attacker could modify the keystroke separation to be greater than or equal to 80 ms; however, an attacker usually will not have the luxury of time and so they will most likely opt for a smaller keystroke separation setting such as *s* = 10 ms. The detection model found in this research assumes that the attacker will opt for smaller keystroke separation timings instead of choosing timings greater than or equal to 80 ms.

As a means of testing the effectiveness of the detection model, two injection devices were used. The first was a Raspberry Pi Zero W with a custom OS image (P4wnP1 A.L.O.A.) that supports injection with various values of *s.* The second, more inconspicuous device, was a Bash Bunny from Hak5. During primary testing with the Raspberry Pi Zero W, a series of tests were run for *s* ≅10 ms, 20 ms, 30 ms, …, 80 ms. The value of *s* and the number of allowed keystrokes, denoted as *k*, were found to be inversely proportional for values of *s* ≅10 ms, 20 ms, …, 50 ms with the maximum value of *k* ≅30.9 and the minimum value being *k*≅11.5. There was a significant number of false negatives for values of *s* ≅ 60 ms, 70 ms, and 80 ms that likely resulted from the decision to take the average value of *s* for four keystrokes and compare this value to the detection threshold of 80 ms. During secondary testing with the Bash Bunny, the detection model showed significant promise with an average value of *k* = 10.8, with *s* ≅10 ms. The results of the primary and secondary tests have two significant implications regarding the detection model implemented in this research. Firstly, a value of *k*≅30.9 might be alarming, but it is important to remember that this includes both whitespace characters found within the payload and the keystrokes needed to launch a terminal environment within a given OS. In addition, some popular Linux-based reverse shell payloads [3] are themselves longer than 30 characters. Second, an attacker is more likely to choose a Bash Bunny for a social engineering engagement due to its inconspicuous nature. Therefore, due to the promising results of secondary testing, the detection/prevention model was discovered to be effective in a practical sense.

**References**

1. Neuner, S., Voyiatzis, A. G., Fotopoulos, S., Mulliner, C., & Weippl, E. R.: USBlock: Blocking USB-based Keypress Injection Attacks.(2018, July 10).
2. Umphress, D., Williams, G.: Identity verification through keyboard characteristics.International journal of man-machine studies 23(3), 263–273 (1985)
3. S. (2020, November 14). Reverse Shell Cheat Sheet. Retrieved December 16, 2020, from https://github.com/swisskyrepo/PayloadsAllTheThings/blob/master/Methodology%20and%20Resources/Reverse%20Shell%20Cheatsheet.md