Lock Keys

Overview

Computer keyboards are typically thought of as being essentially one-way communications peripherals, but this isn't always the case. There are actually methods for bi-directional communications, which may be taken advantage of using the USB Rubber Ducky.

Brief History

First, a brief history. In 1981 the "IBM Personal Computer" was introduced — the origins of the ubiquitous "PC" moniker. It featured an 83-key keyboard that was unique in the way it handled three significant keys.

Caps lock, num lock and scroll lock. Collectively, the lock keys. These toggle keys typically change the behavior of subsequent keypresses. As an example, pressing the caps lock key would make all letter keypresses uppercase. The lock key state would be indicated by a light on the keyboard.

At the time, the 1981 IBM-PC keyboard itself was responsible for maintaining the state of the lock keys and lighting the corresponding LED indicators. With the introduction of the IBM PC/AT in 1984, that task became the responsibility of the computer.

This fundamental change in computer-keyboard architecture carried over from early 1980's and 1990's keyboards, with their DIN and PS/2 connectors, to the de facto standard 104+ key keyboards of the modern USB era.

End Points and Control Codes

Today, keyboards implement the Human Interface Device (USB HID) specification. This calls for an "IN endpoint" for the communication of keystrokes from the keyboard to the computer, and an "OUT endpoint" for the communication of lock key LED states from the computer to the keyboard.

A set of HID codes for LED control (spec code page 08) define this communication. Often, these control codes are sent from the computer to the keyboard via the OUT endpoint when a computer starts. As an example, many computer BIOS (or EUFI) provide an option to enable num lock at boot. If enabled, the control code is sent to the keyboard when the computer powers on.

As another example, one may disable a lock key all together. On a Linux system, command line tools like xmodmap, setxkbmap and xdotool may be used to disable caps lock. Similarly, an edit to registry may perform a similar task on Windows systems.

In both cases the keyboard, naive to the attached computer's configuration, will still send the appropriate control code to the IN endpoint when the caps lock key is pressed. However, the computer may disregard the request and neglect to send the corresponding LED indication control code back to the keyboard via the OUT endpoint.

Synchronous Reports

As demonstrated, a target may accept keystroke input from multiple HID devices. Put another way, all USB HID keyboard devices connected to a computer feature an IN endpoint, from which keystrokes from the keyboard may be sent to the target computer.

Similarly, all USB HID keyboards connected to the computer feature an OUT endpoint, to which the computer may send caps lock, num lock and scroll lock control codes for the purposes of controlling the appropriate lock key LED light.

This may be validated by connecting multiple USB keyboards to a computer. Press the caps lock key on one keyboard, and watch the caps lock indicator on all keyboards light up.

Due to the synchronous nature of the control code being sent to all USB HID OUT endpoints, the USB Rubber Ducky may perform systematic functions based on the state of the lock keys.

WAIT_FOR Commands

The various `WAIT_FOR...` commands will pause payload execution until the desired change occurs, similar to the function of `WAIT_FOR_BUTTON_PRESS`.

Command	Description
`WAIT_FOR_CAPS_ON`	Pause until caps lock is turned on
`WAIT_FOR_CAPS_OFF`	Pause until caps lock is turned off
`WAIT_FOR_CAPS_CHANGE`	Pause until caps lock is toggled on or off
`WAIT_FOR_NUM_ON`	Pause until num lock is turned on
`WAIT_FOR_NUM_OFF`	Pause until num lock is turned off
`WAIT_FOR_NUM_CHANGE`	Pause until num lock is toggled on or off
`WAIT_FOR_SCROLL_ON`	Pause until scroll lock is turned on
`WAIT_FOR_SCROLL_OFF`	Pause until scroll lock is turned off
`WAIT_FOR_SCROLL_CHANGE`	Pause until scroll lock is toggled on or off

Example

```
REM Example WAIT_FOR_CAPS_CHANGE Payload

ATTACKMODE HID STORAGE
LED_OFF
DELAY 2000

WHILE TRUE
    LED_R
    WAIT_FOR_CAPS_CHANGE
```

```
    LED_G
    WAIT_FOR_CAPS_CHANGE
END_WHILE
```

Result

Pressing the caps lock key on the target will cycle the USB Rubber Ducky LED between red and green.

SAVE and RESTORE Commands

The currently reported lock key states may be saved and later recalled using the `SAVE_HOST_KEYBOARD_LOCK_STATE` and `RESTORE_HOST_KEYBOARD_LOCK_STATE` commands.

Example

```
REM Example SAVE and RESTORE of of the Keyboard Lock State

ATTACKMODE HID STORAGE
DELAY 2000

SAVE_HOST_KEYBOARD_LOCK_STATE

$_RANDOM_MIN = 1
$_RANDOM_MAX = 3

VAR $TIMER = 120
WHILE ($TIMER > 0)
    VAR $A = $_RANDOM_INT
    IF ($A == 1) THEN
        CAPSLOCK
    ELSE IF ($A == 2) THEN
        NUMLOCK
    ELSE IF ($A == 3) THEN
        SCROLLLOCK
    END_IF
    DELAY 50
```

```
    $TIMER = ($TIMER - 1)
END_WHILE

RESTORE_HOST_KEYBOARD_LOCK_STATE
```

Result

At the beginning of the payload, the currently reported keyboard lock state are saved.

For about 6 seconds, as a while loop iterates 120 times with a 50 ms delay, the caps, num or scroll lock keys will be randomly pressed.

When the "keyboard fireworks" display has concluded, the previously saved keyboard lock state will be restored.

Meaning, if the target has caps lock off, scroll lock off, and num lock on before the payload began, so too would it after its conclusion.

Internal Variables

The following internal variables relate to the lock keys and may be used in your payload for advanced functions.

Internal Variable	Description
`$_CAPSLOCK_ON`	`TRUE` if on, `FALSE` if off.
`$_NUMLOCK_ON`	`TRUE` if on, `FALSE` if off.
`$_SCROLLLOCK_ON`	`TRUE` if on, `FALSE` if off.
`$_SAVED_CAPSLOCK_ON`	On USB attach, sets `TRUE` or `FALSE` depending on the reported OS condition.
`$_SAVED_NUMLOCK_ON`	On USB attach, sets `TRUE` or `FALSE` depending on the reported OS condition.
`$_SAVED_SCROLLLOCK_ON`	On USB attach, sets `TRUE` or `FALSE` depending on the reported OS condition.
`$_RECEIVED_HOST_LOCK_LED_REPLY`	On receipt of any lock state LED control code, sets `TRUE`. This flag is helpful for fingerprinting certain operating systems (e.g. macOS) or systems which do not communicate lock keys "correctly".

$_RECEIVED_HOST_LOCK_LED_REPLY

```
REM Example Blink green if LED states are reported, otherwise bl

ATTACKMODE HID STORAGE
DELAY 2000

FUNCTION BLINK_RED()
    WHILE TRUE
        LED_OFF
        DELAY 50
        LED_R
        DELAY 50
    END_WHILE
END_FUNCTION

FUNCTION BLINK_GREEN()
```

```
    WHILE TRUE
        LED_OFF
        DELAY 50
        LED_G
        DELAY 50
    END_WHILE
END_FUNCTION

IF ($_RECEIVED_HOST_LOCK_LED_REPLY == TRUE) THEN
    BLINK_GREEN()
ELSE IF ($_RECEIVED_HOST_LOCK_LED_REPLY == FALSE) THEN
    BLINK_RED()
END_IF
```

Result

The USB Rubber Ducky will blink green if the LED states are reported by the target. Otherwise, the LED will blink red.

$_CAPSLOCK_ON

```
REM Example ONLY CAPS FOR YOU (Evil Prank)

ATTACKMODE HID STORAGE
DELAY 2000

WHILE TRUE
    IF ($_CAPSLOCK_ON == FALSE) THEN
        CAPSLOCK
    END_IF
    DELAY 100
END_WHILE
```

Result

If caps lock is turned off by the user, it will be turned on by the USB Rubber Ducky.