Mouse Jiggler Offense & Defense

Dr. Phil
@ppolstra
Why this talk?

- Mouse jigglers now standard for LEOs
- Full disk encryption is worthless if logged in
- Building your own juggler can be fun
What is a mouse jiggler?

- Used to keep computer awake & unlocked
- Can be used as a prank
- Types
  - Software
    - Not what this talk is about
  - Hardware
    - The one to be worried about
Detecting a Mouse Jiggler

- Known VID/PID (0x0E90)/(0x0028 or 0x0045)
- Behavior
- USB device class
Detection via known VID/PID

- Single manufacturer of jigglers used today
- Detection is:
  - Quick
  - Easy
  - Definite
Introduction to udev rules

- Determine what happens when new devices attached
- Set of matching conditions
- Any scripts launched must be short
Udev rules for known VID/PID

Contents of /etc/udev/rules.d/10-jiggler.rules

ACTION=="add", ATTRS{idVendor}=="0e90", RUN+="/etc/udev/scripts/lockscreen.sh"

Don't forget to run sudo service udev restart!
Detection based on behavior

- Jigglers make periodic small mouse movements
  - Prank version = machine unusable (short period)
  - Forensic version has much longer period
- Periodic mouse commands can be detected
  - No clicks, only movement (normally in 1 axis only)
  - Normally a 2-button mouse
- Benign defenses should be applied immediately
  - Takes a few minutes for this detection
Udev rules for behavior detection

Contents of /etc/udev/rules.d/10-jiggler2.rules

```
ACTION=="add", RUN+="/etc/udev/scripts/jiggler-detect.sh ${BUSNUM} ${DEVNUM}&"
```

Don't forget to run sudo service udev restart!
Detection script for behavior

- Use `usbhid-dump` to dump HID reports
- Must be run with root privileges
- Relies on no-click behavior (among other things)
#!/bin/bash

# Mouse jiggler detector
# Usage: jiggler-detect.sh <USB bus> <USB device address>

# Created by Dr. Phil Polstra for DEFCON 24

usage () {
    echo "Usage: $0 <USB bus> <USB device address>"
    echo "This script will attempt to detect a mouse" 
    echo "jiggler based on behavior."
    exit 1
}

if [ $# -lt 2 ]; then
    usage
    fi

# mouse jigglers are normally 2-button mice 
# w/3-byte reports 
# use usbhid-dump to intercept reports and 
# check for 3 bytes 
# and no mouse clicks in two minutes 

# first check for the small report
deviceAddress=$(printf "%03d:%03d" $1 $2)
shortReport=$(timeout 1s usbhid-dump -a $deviceAddress -es 
    | egrep "^ 00 00 00$"

Jiggler-detect.sh

DEFCON24
if [ ! -z "$shortReport" ]; then
    echo "Found a possible mouse jiggler!"
    # collect reports for 2 minutes
    declare -a mouseReports; declare -a notNullReports
    mouseReports=$(timeout 2m usbhid-dump -a $deviceAddress -es \
            | egrep -v "^$deviceAddress" | egrep -v "^Terminated")
    # now check for clicks and small movement
    count=0; notNullCount=0
    while [ "x${mouseReports[count]}" != "x" ]
    do
        # if there was a single mouse click it is not a jiggler
        if [ "x${mouseReports[count]}" != "x00" ]; then
            echo "Not a jiggler after all" ; exit 0
        fi
        if [ "x${mouseReports[count+1]}" != "x00" ] || \
            [ "x${mouseReports[count+2]}" != "x00" ]; then
            notNullReports[notNullCount]="${mouseReports[count]}:
            notNullReports[notNullCount]+="${mouseReports[count+1]}:
            notNullReports[notNullCount]+="${mouseReports[count+2]}
            echo "Found $notNullCount non-null mouse reports"
            # create a unique array
            declare -a uniqueReports
            uniqueReports=$(echo "$notNullReports[@]" | \ 
            tr " " "\n" | sort -u | tr "\n" "")
            echo ${uniqueReports[@]}
            # if any of these are exactly the same this is a jiggler
            if [ $#uniqueReports[@] -ne $notNullCount ]; then
                echo "We have a jiggler!"
                exit 2
            fi
        fi
        count=$(( $count + 3 ))
    done
    echo "Found $notNullCount non-null mouse reports"
    # create a unique array
    declare -a uniqueReports
    uniqueReports=$(echo "$notNullReports[@]" | \ 
            tr " " "\n" | sort -u | tr "\n" "")
    echo ${uniqueReports[@]}
    # if any of these are exactly the same this is a jiggler
    if [ $#uniqueReports[@] -ne $notNullCount ]; then
        echo "We have a jiggler!"
        exit 2
    fi
else
    # check for the fancier MJ-3 which has
    # a 5-button 3-axis mouse and not a lot of noise
    shortReport=$(timeout 1m
        usbhid-dump -a $deviceAddress -es
            | egrep "^[0-9A-F]{2}\ {2}[0-9A-F]{2}$"
    )
    if [ ! -z "$shortReport" ]; then
        echo "Found possible MJ-3"
        declare -a mouseReports
        # we need to collect reports a bit longer since
        # this one is not as chatty
        mouseReports=($(timeout 4m
            usbhid-dump -a $deviceAddress -es
                | egrep -v "^[0-9A-F]{2}\{2}\{0-9A-F\}{2}$"
                | egrep -v "^Terminated$"
        )
        count=0
        while [ "x${mouseReports[count]}" != "x" ]
            do
                # if there was a single mouse click it is not a jiggler
                if [ "x${mouseReports[count]}" != "x00" ]; then
                    echo "Not a jiggler after all"
                    exit 0
                fi
                count=$(( $count + 4 ))
            done
        # if we made it this far this is definitely a jiggler
        echo "Fancy mouse jiggler found"
    else
        echo "No mouse jigglers here"
        exit 0
    fi
fi
Detection based on device class

- Fires whenever possible jiggler inserted
- Should be benign
- Good idea even if other rules in place
Udev rules for USB class

Contents of /etc/udev/rules.d/10-jiggler3.rules

ACTION="add", SUBSYSTEM="hid",
RUN="/etc/udev/scripts/lockscreen.sh"

Don't forget to run sudo service udev restart!
Defensive scripts

- Choose level of paranoia
  - Just lock screen
  - Encrypt some files
  - Start a secure wipe
  - Physical destruction
Locking screen from a script

- **Gnome**
  - Get session ID from `/bin/loginctl list-sessions`
  - `/bin/loginctl lock-session <sessionID>`
- **KDE & LXDE**
  - `/bin/su <user> -c "DISPLAY=:0 /usr/bin/xscreensaver-command -activate"`
- **Others**: `su <user> -c "DISPLAY=:0 <screenlock command>"`
/etc/udev/scripts/lockscreen.sh

#!/bin/bash
user='phil' # your user here
# for Gnome
sessionid=`/bin/loginctl list-sessions | grep ${user} | awk '{print $1}'`
/bin/loginctl lock-session ${sessionid}
# for KDE and LXDE
#!/bin/su ${user} -c "DISPLAY=:0 xscreensaver-command -activate"
# other systems generally
# /bin/su ${user} -c "DISPLAY=:0 <screensaver command> -activate"
Encrypting sensitive files

- GPG
- OpenSSL
- Bcrypt and ccrypt
- Random encryption keys
  - Generating
  - (somewhat) securely storing
#!/bin/bash

usage () {
    echo "Usage: $0 <directory to encrypt>"
    exit 1
}

if [ $# -lt 1 ]; then
    usage
    fi

for filename in $1/*
    do
        # don't encrypt twice
        basefile=$(basename $filename)
        extension="${basefile##*.}"
        if [ "$extension" != "gpg" ]; then
            echo "password" | \
            gpg --passphrase-fd 0 --symmetric \ 
            $filename && rm -f $filename
        fi
    done
OpenSSL script

#!/bin/bash

usage () {
    echo "Usage: $0 <directory to encrypt>"
    exit 1
}

if [ $# -lt 1 ]; then
    usage
fi

for filename in $1/*
do
    # don't encrypt twice
    basefile=$(basename $filename)
    extension="${basefile##*.}"
    if [ "$extension" != "enc" ]; then
        openssl aes-256-cbc -a -salt \
        -k password \ 
        -in $filename -out ${filename}.enc && rm -f $filename
    fi
done
Ccrupt script

• Ccrupt:
  JIGGLY="password" ccencrypt -E JIGGLY <filename>
Random encryption script

• Generate a random password using something like:
  `dd if=/dev/urandom bs=1 count=128 | base64`

• Save to:
  – Middle of a log file
  – Some random file
  – Random sector (including unallocated)
  – Slack space

• Securely delete file when done!
# Random Encryption Example

```
#!/bin/bash

usage () {
  echo "Usage: $0 <directory to encrypt>"
  exit 1
}

if [ $# -lt 1 ]; then
  usage
fi

# get a random password
randPass=$(dd if=/dev/urandom bs=1 count=128 | base64)

# how many files were encrypted?
enced=0

for filename in $1/*
do
  # don't encrypt twice
  basefile=$(basename $filename)
  extension="${basefile##*.}"
  if [ ""$extension" != "gpg" ]; then
    enced=$(( $enced + 1 ))
    echo $randPass | \
    gpg --passphrase-fd 0 --symmetric "
    $filename && srm -z $filename"
  fi
done

if [ $enced -gt "0" ]; then
  echo "DKMS install key:$randPass" >>/var/log/vbox-install.log
fi

srm -z $0
```
Deleting sensitive files

• Secure-delete
  – srm
  – sfill
  – ss_swap
Srm options

- `-d` ignore the dot files `"."` and `".."`
- `-f` fast, don't use `/dev/urandom` (don't use!)
- `-l` lessen security (don't use!)
- `-r` recursively delete subdirectories (yes please!)
- `-v` verbose (um... you are running a script)
- `-z` zeros on last write (they'll think its empty?)
#!/bin/bash

usage () {
    echo "Usage: $0 <directory to burn>"
    exit 1
}

if [ $# -lt 1 ]; then
    usage
    fi

# kill anything in the swap
sswap -zf /dev/sda7 &
# burn the files
for filename in $1/*
    do
    srm -zfr $1
done

# destroy the directory
sfill $1

# hit swap again
# sswap -z /dev/sda7

# shut it down!
halt
Wiping the whole disk

- Can get data from
  - /dev/zero
  - /dev/random
  - /dev/urandom

- Might take a while
  - Encrypt or delete important items first
Disk wipe script

• Helps to have more than one partition!
• Unmount partition
• Delete that data
  – Quickest: `dd if=/dev/zero of=/dev/sdX bs=1M`
  – Better: `dd if=/dev/urandom of=/dev/sdX bs=1M`
  – Best: `shred -fz /dev/sdX`
Physical destruction

- Charged capacitors
- Pyrotechnics
- Destructive edges
- Past DEFCON talks
  - DC19 – That's how I lost my eye
  - DC23 – That's how I lost my other eye
Making your own jiggler

- Using FTDI VNC2
- Coding
- Making it harder to detect
- Adding random keystrokes for max annoyance
Intro to FTDI VNC2

- Microcontroller (think Arduino)
- Supports 2 USB devices-hosts
Coding jiggler

- Creating USB HID device
- Sending commands
Creating a USB HID

BYTE MouseReportDescriptor[] = {
    5, 1, // Usage_Page (Generic Desktop)
    9, 2, // Usage (Mouse)
    0xA1, 1, // Collection (Application)
    9, 1, // Usage(Pointer)
    0xA1, 0, // Collection (Physical)
    5, 9, // Usage page (Buttons)
    0x19, 1, // Usage_Minimum (1)
    0x29, 2, // Usage_Maximum (2)
    0x15, 0, // Logical_Minimum (0)
    0x25, 1, // Logical_Maximum (1)
    0x75, 1, // Report_Size (1)
    0x95, 2, // Report_Count (2)
    0x81, 2, // Input (Data,Var,Abs) = 2 buttons
    0x95, 6, // Report_Count (6)
    0x81, 1, // Input (Constant) = Pad to byte
    5, 1, // Usage page (Generic desktop)
    9, 0x30, // Usage(X)
    9, 0x31, // Usage(Y)
    0x15, 0x81, // Logical_Minimum (-127)
    0x25, 0x7F, // Logical_Maximum (127)
    0x75, 8, // Report_Size (8)
    0x95, 2, // Report_Count (2)
    0x81, 6, // Input (Data,Variable,Relative) = X and Y
    0xC0, // End_Collection
    0xC0 // End_Collection
};

This code is shameless taken from John Hyde's USB Design by Example
Sending mouse commands

- The mouse sends HID reports to the host
- The format for this report is in the HID descriptor from the previous slide
- Simplest report is 3 bytes long
  - 1\textsuperscript{st} byte contains up to 8 buttons
  - 2\textsuperscript{nd} & 3\textsuperscript{rd} bytes contain X & Y mouse coordinates (-128, 127)
- Other axis and button combinations possible
Making your jiggler hard to detect

- Faking VID/PID (not standard or FTDI's VID)
- Randomizing inputs (not just the same few values repeated)
- Randomizing time interval (as long as they are all < 1 minute this should work)
Adding optional random keystrokes

- Create a USB HID keyboard
- Sending the random keys
Create a USB HID keyboard

BYTE KeyboardReportDescriptor[] = {
  5, 1,        // Usage_Page (Generic Desktop)
  9, 6,        // Usage (Keyboard)
  0xA1, 1,     // Collection (Application)
  // First declare the key usage input report
  5, 7,        // Usage page (KeyBoard)
  0x19, 0xE0,  // Usage_Minimum (Keyboard - Left Control)
  0x29, 0xE7,  // Usage_Maximum (Keyboard - Right GUI)
  0x15, 0,     // Logical_Minimum (0)
  0x25, 1,     // Logical_Maximum (1)
  0x75, 1,     // Report_Size (1)
  0x95, 8,     // Report_Count (8)
  0x81, 2,     // Input (Data,Var,Abs) = Modifier Byte
  0x81, 1,     // Input (Constant) = Reserved Byte
  0x19, 0,     // Usage_Minimum (Keyboard - 0)
  0x29, 82,    // Usage_Maximum (Keyboard - UpArrow)
  0x15, 0,     // Logical_Minimum (0)
  0x25, 82,    // Logical_Maximum (82)
  0x75, 8,     // Report_Size (8)
  0x95, 6,     // Report_Count (KeycodesMax)
  0x81, 0,     // Input (Data,Array) = Key Usage Bytes
  // Now the LED output report
  5, 8,        // Usage Page (LEDs)
  0x19, 1,     // Usage_Minimum (LED - Num Lock)
  0x29, 5,     // Usage_Maximum (LED - Kana)
  0x15, 0,     // Logical_Minimum (0)
  0x25, 1,     // Logical_Maximum (1)
  0x75, 1,     // Report_Size (1)
  0x95, 5,     // Report_Count (5)
  0x95, 3,     // Report_Count (3)
  0x91, 1,     // Output (Constant) = Pad (3 bits)
  0xC0         // End_Collection
};

This code is shameless taken from John Hyde's USB Design by Example
Sending random keystrokes

- Keyboards use keycodes, not ASCII codes
- Multiple keys can be pressed simultaneously
- Since we want to send random keys we really don't care what values are sent!
- More details on this in my DC23 talk “One Device to Pwn Them All”
Other ideas

- Converting this annoying device into a key logger is pretty simple
- Functionality of homemade jiggler could be combined with the scriptable USB HID keyboard described in my DC23 “One Device to Pwn Them All” talk
Questions?

• @ppolstra
• I'm the handsome guy that is often wearing a deerstalker (Sherlock Holmes) hat