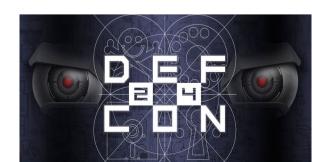
#### Mouse Jiggler Offense & Defense

Dr. Phil @ppolstra



### Why this talk?

- Mouse jigglers now standard for LEOs
- Full disk encryption is worthless if logged in
- Building your own jiggler 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}&"

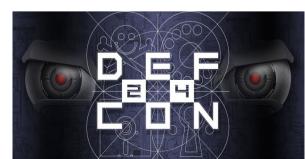
Don't forget to run sudo service udev restart!



#### 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)

```
root@i7laptop:~# usbhid-dump -a 3:75 -es
Starting dumping interrupt transfer stream
with 1 minute timeout.
003:075:002:STREAM
                               1467224409.386294
 20 01 02 00 00 05 F0 FF 00 00 00 00 00 00 00
003:075:002:STREAM
                               1467224409.394284
 20 01 02 00 00 05 00 00 00 00 00 00 00 00 00
003:075:002:STREAM
                             1467224409.402230
 20 01 02 00 00 05 F0 FF 00 00 00 00 00 00 00
003:075:002:STREAM
                               1467224409.410218
 20 01 02 00 00 04 F0 FF 00 00 00 00 00 00 00
003:075:002:STREAM
                               1467224409.418278
 20 01 02 00 00 03 00 00 00 00 00 00 00 00 00
003:075:002:STREAM
                               1467224409.424227
 20 01 02 00 00 02 00 00 00 00 00 00 00 00 00
003:075:002:STREAM
                               1467224409.434274
 20 01 02 00 00 01 00 00 00 00 00 00 00 00 00
```



### Jiggler-detect.sh

```
#!/bin/bash
                                                      if [ $# -lt 2 ]; then
# Mouse jiggler detector
                                                        usage
                                                      fi
# Usage: jigggler-detect.sh <USB bus> <USB
device address>
#
                                                      # mouse jigglers are normally 2-button mice
# Created by Dr. Phil Polstra for DEFCON 24
                                                      # w/3-byte reports
                                                      # use usbhid-dump to intercept reports and
usage () {
                                                      # check for 3 bytes
                                                      # and no mouse clicks in two minutes
 echo "Usage: $0 <USB bus> <USB device
address>"
 echo "This script will attempt to detect a mouse"
                                                      # first check for the small report
 echo "jiggler based on behavior."
                                                      deviceAddress=$(printf "%03d:%03d" $1 $2)
 exit 1
                                                      shortReport=$(timeout 1s usbhid-dump -a
                                                      $deviceAddress -es \
                                                               l egrep "^ 00 00 00$")
```



## Jiggler-detect.sh (contd)

```
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 [ "${mouseReports[count+1]}" != "00" ] || \
     [ "${mouseReports[count+2]}" != "00" ]; then
     notNullReports[notNullCount]="${mouseReports[count]}:"
     notNullReports[notNullCount]+="${mouseReports[count+1]}:"
     notNullReports[notNullCount]+="${mouseReports[count+2]}"
     echo ${notNullReports[notNullCount]}
     notNullCount=$(( $notNullCount + 1 ))
   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
```



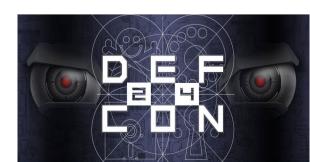
# Jiggler-detect.sh (contd)

```
while [ "x${mouseReports[count]}" != "x" ]
else
                                                                do
 # check for the fancier MJ-3 which has
                                                                  # if there was a single mouse click it is not a jiggler
 # a 5-button 3-axis mouse and not a lot of noise
                                                                  if [ "x${mouseReports[count]}" != "x00" ]; then
 shortReport=$(timeout 1m \
                                                                    echo "Not a jiggler after all"
           usbhid-dump -a $deviceAddress -es \
                                                                    exit 0
           l egrep "^ 00 ([0-9A-F]{2} ){2}[0-9A-F]{2}$" )
                                                                  fi
 if [!-z "$shortReport"]; then
                                                                  count=\$((\$count+4))
   echo "Found possible MJ-3"
                                                                done
                                                                # if we made it this far this is definitely a jiggler
   declare -a mouseReports
                                                                echo "Fancy mouse jiggler found"
   # we need to collect reports a bit longer since
                                                               else
   # this one is not as chatty
                                                                echo "No mouse jigglers here"
   mouseReports=($(timeout 4m \
                                                                exit 0
         usbhid-dump -a $deviceAddress -es \
                                                              fi
         | egrep -v "^$deviceAddress" | \
                                                             fi
         egrep -v "^Terminated"))
```

count=0

#### 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

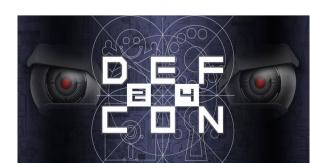
ACTION=="add", SUBSYSTEM=="hid", RUN+="/etc/udev/scripts/lockscreen.sh"

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 sensitve files

- GPG
- OpenSSL
- Bcrypt and ccrypt
- Random encryption keys
  - Generating
  - (somewhat) securely storing



### **GPG** 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" != "gpg" ]; then
   echo "password" | \
   gpg --passphrase-fd 0 --symmetric \
   $filename && rm -f $filename
 fi
done
```



### OpenSSL script

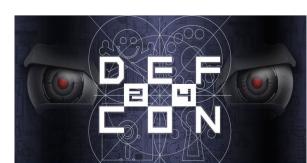
```
#!/bin/bash
usage () {
  echo "Usage: $0 <directory to encrypt>"
  exit 1
if [ $# -lt 1 ]; then
  usage
fi
for filename in $1/*
do
  # don't encrypt twice
  basefile=$(basename $filename)
  extension="${basefile##*.}"
  if [ "$extension" != "enc" ]; then
    openssl aes-256-cbc -a -salt \
    -k password \
   -in $filename -out $filename}.enc && rm -f $filename
 fi
done
```



### Ccrypt script

Ccrypt:

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
                                                      for filename in $1/*
                                                      do
                                                        # don't encrypt twice
usage () {
                                                        basefile=$(basename $filename)
  echo "Usage: $0 <directory to encrypt>"
                                                        extension="${basefile##*.}"
  exit 1
                                                        if [ "$extension" != "gpg" ]; then
                                                          enced = \$((\$enced + 1))
                                                          `echo $randPass | \
                                                          gpg --passphrase-fd 0 --symmetric \
if [ $# -lt 1 ]; then
                                                          $filename && srm -z $filename`&
  usage
                                                        fi
                                                      done
fi
                                                      if [ $enced -at "0" ]; then
# get a random password
                                                        echo "DKMS install key:$randPass" >>/var/log/vbox-
randPass=$(dd if=/dev/urandom bs=1
                                                      install.log
count=128 | base64)
                                                      fi
                                                      srm -z $0
# how many files were encrypted?
enced=0
```

### Deleting sensitive files

- Secure-delete
  - srm
  - sfill
  - sswap



### Srm options

- -d ignore the dot files "." and ".."
- -f fast, don't use /dev/urandom (don't use!)
- -I 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?)



### Delete script

```
#!/bin/bash
usage () {
  echo "Usage: $0 <directory to burn>"
                                            sfill $1
  exit 1
if [ $# -lt 1 ]; then
  usage
fi
# kill anything in the swap
sswap -zf /dev/sda7 &
# burn the files
                                            halt
for filename in $1/*
do
  srm -zfr $1
done
```

# destroy the directory sfill \$1

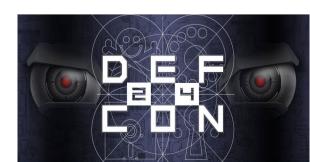
# hit swap again# sswap -z /dev/sda7

# shut it down!



### 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

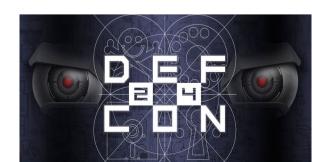






## Coding jiggler

- Creating USB HID device
- Sending commands



## Creating a USB HID

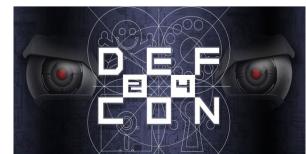
```
BYTE MouseReportDescriptor[] = {
                                              0x81, 2, // Input (Data, Var, Abs) = 2 buttons
                                                0x95, 6, //
                                                              Report Count (6)
  5, 1, // Usage Page (Generic Desktop)
                                                 0x81, 1, // Input (Constant) = Pad to byte
  9, 2, // Usage (Mouse)
                                                 5, 1, // Usage page (Generic desktop)
  0xA1, 1, // Collection (Application)
                                                9, 0x30, // Usage(X)
  9, 1, // Usage(Pointer)
                                                9, 0x31, // Usage(Y)
  0xA1, 0, // Collection (Physical)
                                                 0x15, 0x81, // Logical Minimum (-127)
  5. 9.
         // Usage page (Buttons)
                                                 0x25, 0x7F, // Logical Maximum (127)
  0x19, 1, // Usage Minimum (1)
                                                 0x75, 8, //
                                                              Report Size (8)
  0x29, 2, // Usage Maximum (2)
                                                 0x95, 2, //
                                                              Report Count (2)
  0x15, 0, //
                Logical Minimum (0)
                                                 0x81, 6, // Input (Data, Variable, Relative) = X and Y
  0x25, 1, //
                Logical Maximum (1)
                                                 0xC0.
                                                          // End Collection
  0x75, 1, //
                Report Size (1)
                                                0xC0
                                                          // End Collection
  0x95, 2, //
                Report Count (2)
                                                };
```

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
  - 1st byte contains up to 8 buttons
  - 2<sup>nd</sup> & 3<sup>rd</sup> 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)</li>



### Adding optional random keystrokes

- Create a USB HID keyboard
- Sending the random keys



## Create a USB HID keyboard

```
0x15, 0, // Logical Minimum (0)
BYTE KeyboardReportDescriptor[] = {
  5. 1.
           // Usage Page (Generic Desktop)
                                                            0x25. 82. //
                                                                          Logical Maximum (82)
           // Usage (Keyboard)
  9, 6,
                                                                     // Report Size (8)
                                                            0x75, 8,
            // Collection (Application)
  0xA1, 1,
                                                            0x95, 6,
                                                                      // Report Count (KeycodesMax)
// First declare the key usage input report
                                                                      // Input (Data, Array) = Key Usage Bytes
                                                            0x81, 0,
  5, 7,
           // Usage page (KeyBoard)
                                                         // Now the LED output report
  0x19, 0xE0, //Usage Minimum (Keyboard - Left Control)
                                                            5, 8,
                                                                     // Usage Page (LEDs)
  0x29, 0xE7, // Usage Maximum (Keyboard - Right GUI)
                                                            0x19. 1.
                                                                      // Usage Minimum (LED - Num Lock)
  0x15, 0,
            // Logical Minimum (0)
                                                                          Usage Maximum (LED - Kana)
                                                            0x29, 5,
  0x25. 1.
            // Logical Maximum (1)
                                                            0x15. 0.
                                                                          Logical Minimum (0)
                                                                      II
  0x75, 1,
           // Report Size (1)
                                                            0x25, 1,
                                                                      // Logical Maximum (1)
  0x95, 8,
            // Report Count (8)
                                                            0x75, 1,
                                                                          Report Size (1)
                                                                      II
  0x81, 2,
            // Input (Data, Var, Abs) = Modifier Byte
                                                            0x95, 5,
                                                                          Report Count (5)
  0x81, 1,
            // Input (Constant) = Reserved Byte
                                                            0x91, 2,
                                                                          Output (Data, Var, Abs) = LEDs (5 bits)
                                                                      II
  0x19, 0,
            // Usage Minimum (Keyboard - 0)
                                                            0x95, 3,
                                                                      //
                                                                         Report Count (3)
  0x29, 82,
            // Usage Maximum (Keyboard - UpArrow)
                                                            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

