DUCKY SCRIPT - THE USB RUBBER DUCKY LANGUAGE

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 Windows desktop icons
```

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 is envious of CONTROL
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 jealous of CONTROL's leetness and CTRL+S
```

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
```

OBFUSCATION AND OPTIMIZATION

While this post 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.

```
DELAY 1000
GUI r
DELAY 100
STRING cmd
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.

```
DELAY 500
STRING color FE
ENTER
STRING mode con:cols=18 lines=1
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. Bonus: For 1337 mode, issue "**color a**"

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:

```
STRING tree c:\ /F /A
ENTER
DELAY 20000
STRING exit
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.

```
DELAY 1000
GUI r
DELAY 100
```

```
STRING cmd /C color FE&mode con:cols=18 lines=1&tree c:\ /F
/A
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.

OPTIMIZING THE OPTIMIZED

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.

WRITING YOUR FIRST USB RUBBER DUCKY PAYLOAD

YOUR FIRST PAYLOAD

Writing a successful payload is a process of continuously researching, writing, encoding, testing and optimizing. Often times 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 and encoding tools.

Let's begin by 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 simple type "c:\windows\notepad.exe" then press enter and notepad 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 notepad open we should be able to simple 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.
Target: Windows 95 and beyond. 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.

```
DELAY 1000
```

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

```
STRING Hello World
```

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

```
REM Type Hello World into Windows notepad. Target: Windows 95
and beyond. Author: Darren
DELAY 1000
GUI r
DELAY 100
STRING c:\windows\notepad.exe
ENTER
DELAY 1000
```

```
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. Being an open source project, there are many encoders available on most platform from a range of programming languages. There are even online encoders which will convert your ducky script to an inject.bin without installing any software. This section will cover the basics of encoding a ducky script into an inject.bin file ready for deployment on the USB Rubber Ducky.

There are two primary ways to encode Ducky Script, either with the Java command line encoder, or the JS Ducky Encoder.

JS DUCKY ENCODER

The Javascript Ducky Encoder is a single HTML file which can be run in a modern browser locally to develop and encode inject.bin files. Download the latest from downloads.hak5.org or directly here.

This is the best way to get started encoding ducky script as it will work on most modern operating systems and browsers without the need for additional command line tools.

JAVA BASED COMMAND LINE ENCODER

For advanced users, or those wishing to incorporate the encoder into a script or existing workflow, the java command line encoder may be ideal.

The standard encoder is a cross-platform java command line tool. It has been greatly enhanced by the community, with many contributions from user midnitesnake. Download it from the resources section of usbrubberducky.com and save it in a convenient directory along with your

helloworld.txt ducky script from the previous step. The Java runtime environment is required in order to run the duckencoder.jar file. If Java isn't already installed, it can be found for most operating systems from java.com/download.

From a command prompt, navigate to this directory and run the jar file with java.

```
java -jar duckencoder.jar
```

The usage, arguments and script commands will display. The standard usage is to specify an input file, and output file and optionally a language. Encode the helloworld.txt into an inject.bin with the following:

```
java -jar duckencoder.jar -i helloworld.txt -o inject.bin
```

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 version 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:

```
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! Go forth and duck 'em!

THE USB RUBBER DUCKY 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.

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 machine's 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.