How to run the "Pokémon Plays Twitch" polyglot

On an emulator

You can play the PoC||GTFO 10 polyglot on a modified lsnes with the hybrid emulation core using bsnes and Gambatte. Figure 1 contains some compilation notes, based on a chrooted installation of Debian Stretch. This requires about 3Gb of free space. Skip chroot instructions and adjust paths and users accordingly if you prefer to mess with your primary OS.

And for the lucarnes amateurs, you may download and run `lsnes-rr2-beta23-installer.exe` from TASVideos[1] and install it.

Then you'll need to drop the two required ROMs in the `rom` directory: "Super Game Boy"[2] as `sgb.sfc` and "Pokemon - Red Version (USA_Europe).gb"[3], as well as `pocorgtfo10.pdf` renamed as `pocorgtfo10.lsmv` somewhere in the chroot.

Fire up lsnes with the Gambatte plugin.

```
./lsnes --library=gambatte/core.so
```

Equivalently, under Windows: run `C:\Program Files (x86)\lsnes\lsnes-bsnes.exe`, then File → Load → Load dynamic link library → `core.dll`

Adjust the ROM paths in Configure → Settings → Advanced: set Paths → firmware and Paths → ROMs to the `rom` directory. Open the movie: File → Load → Movie → `pocorgtfo10.lsmv` and select proposed ROMs. Then SGB → Unpause and enjoy!

On real hardware

If you're the happy owner of a Super Nintendo, a Super Game Boy adapter and a Pokémon Red Game Boy cartridge as shown in Figure 3, you might as well run it on real hardware. You'll need some additional hardware; currently, the only hardware that can handle the timing requirements of executing this exploit on a real console is the NES/SNES replay device from true[4] (although it is currently available primarily in kit form). You will also need to build wiring harnesses for two 7-wire controllers, which will likely require three SNES controllers (the third controller used as a donor for extra wires; it's a cheaper option than buying a real MultiTap device just for its cables). Finally, unless you have fast enough reflexes to reset the console within a 20 ms window during the game save sequence it is wise to connect a wire from pin 4 of the ISCP header on the replay device to the reset pin on the SNES (pin 19 on the expansion header on the underside of the console).

Rather than explaining how to use the bot in detail we'll instead defer to the documentation[5] which explains the basics.

In the original presentation there were two r16m files[6]. The first file completed Stage 0 and triggered the reset on the final frame through the use of a specific flag (`ser.write("~W")`, see below.) The second one completed the remaining stages through the end of writing the Twitch chat client.

Because the payload created for this article is the article itself, it is packaged as a third r16m file, which can be played back immediately after the second r16m file to cause the article text to appear in the chat window at maximum speed. If the original Twitch interface is desired, a few extra scripts are required as we'll see later.

We could have included the r16m files in the zip but that's cheating, isn't it? Instead we propose you to extract the r16m files from the lsmv polyglot itself. Take note that this is not for the faint of heart as it requires a fair bit of effort and some stitching due to the bsnes SGB emulation being inaccurate compared to real hardware. You'll need both the `lsnes_dump_frames.lua` and `lsnes_dump_latch_subf.lua` scripts[7].

Start by loading the polyglot in lsnes as a movie as described in the previous section[8]. By the way, at each edge of dumping the movie it is advisable to Save → State... so you can go back to that point later.

Load the first Lua script: Tools → Run Lua script... → `lsnes_dump_frames.lua`. Execute in

[1] `http://tasvideos.org/Lsnes.html`

[2] `SHA-256: BBA9C269273BEDB9B38BD5EB23BFAA6E509B8DECC7CB80BB5513905AF04F4CEB`

[3] `SHA-256: 5CA7BA01642A3B27B0CC0B5349B52792795B62D3ED977E98A09390659AF96B7B`

[4] `http://truecontrol.org`

[5] `https://svn.truecontrol.org/tasbot/nes-snes-replay/trunk/nes-snes-replay.X/readme.txt`

[6] The r16m files consist of a bitstream of each button on each controller to press.

[7] `unzip -j pocorgtfo10.pdf pokemon_plays_twitch/lsnes_dump*.lua`

[8] If you don't have the required ROMs, since you own the real hardware, you can use e.g. savestates to extract them yourself.

```
# cf https://wiki.debian.org/chroot
sudo apt-get install binutils debootstrap
mkdir -p /path/to/chroot-stretch
sudo debootstrap stretch /path/to/chroot-stretch
sudo chroot /path/to/chroot-stretch
cat > ./usr/sbin/policy-rc.d <<EOF
#!/bin/sh
exit 101
EOF
chmod a+x ./usr/sbin/policy-rc.d
dpkg-divert --divert /usr/bin/ischroot.debianutils --rename /usr/bin/ischroot
ln -s /bin/true /usr/bin/ischroot

apt-get install git make gcc g++ g++-4.9
apt-get install zlib1g-dev libswscale-dev portaudio19-dev libao-dev lua5.2 liblua5.2-dev \
    libcurl4-openssl-dev libgcrypt20-dev libwxgtk3.0-dev libboost1.58-dev \
    libboost-iostreams1.58-dev libboost-filesystem1.58-dev
cd
git clone http://repo.or.cz/lsnes.git
mkdir gambatte && cd gambatte
mkdir lsnes-core
# bb5cd617cb396d11415de7a4af6ab170a7d84136 from exp/gambatte-bsnes-sgb
git -C ../lsnes archive bb5cd617cb396d11415de7a4af6ab170a7d84136 | \
    tar -xf - -C lsnes-core
mkdir -p lsnes-core/bsnes
# 8f448c3061eef5ffaae2b2235d22be3453708f75 from exp/bsnes-ext-gb
git -C ../lsnes archive 8f448c3061eef5ffaae2b2235d22be3453708f75 | \
    tar -xf - -C lsnes-core/bsnes
mkdir -p lsnes-core/gambatte
# 705b3154f683a42c245602a9e66b0b6c71e101df from exp/gambatte-sgb
git -C ../lsnes archive 705b3154f683a42c245602a9e66b0b6c71e101df | \
    tar -xf - -C lsnes-core/gambatte
mkdir -p lsnes-core/bsnes/obj lsnes-core/bsnes/out
make -C lsnes-core
cd
cp gambatte/lsnes-core/core.so lsnes/gambatte/ && rm -rf gambatte
cd lsnes
git checkout 610685db0fc4565f25772eaff2ad47e268fd2a41 # just to be sure
git archive d39571de650d49636778a73c66414aff372c08af | tar -xf - -C bsnes
sed -i 's/^LUA=.*/LUA=lua5.2/' options.build
make
mkdir rom
```

Figure 1 – Compilation notes for Debian Stretch in Chroot

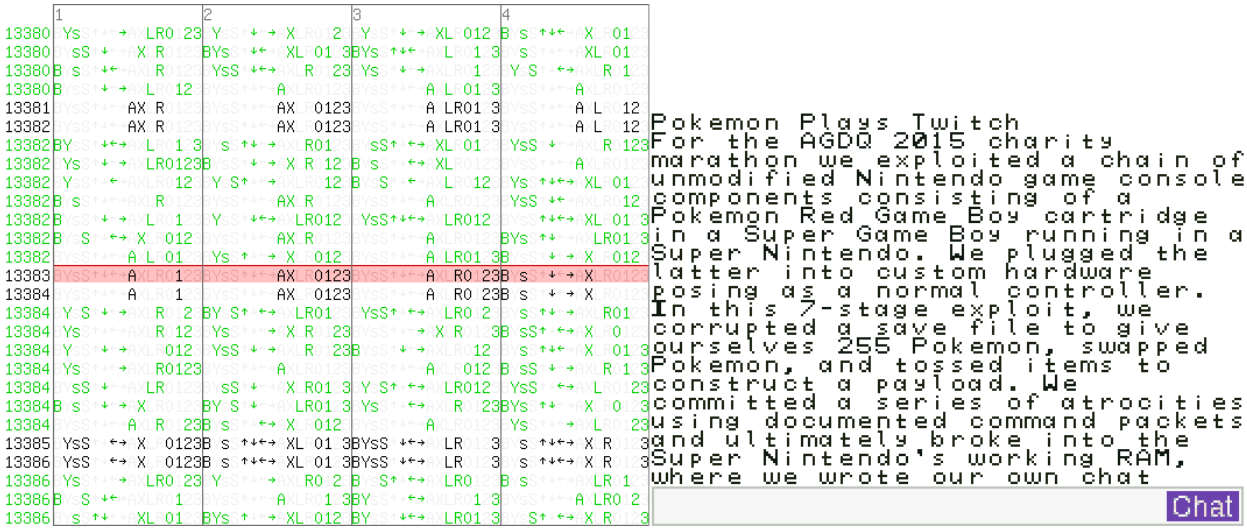

Figure 2 – PoC||GTFO 10:3 displayed in lsnes, with 4 controllers using sub-frames

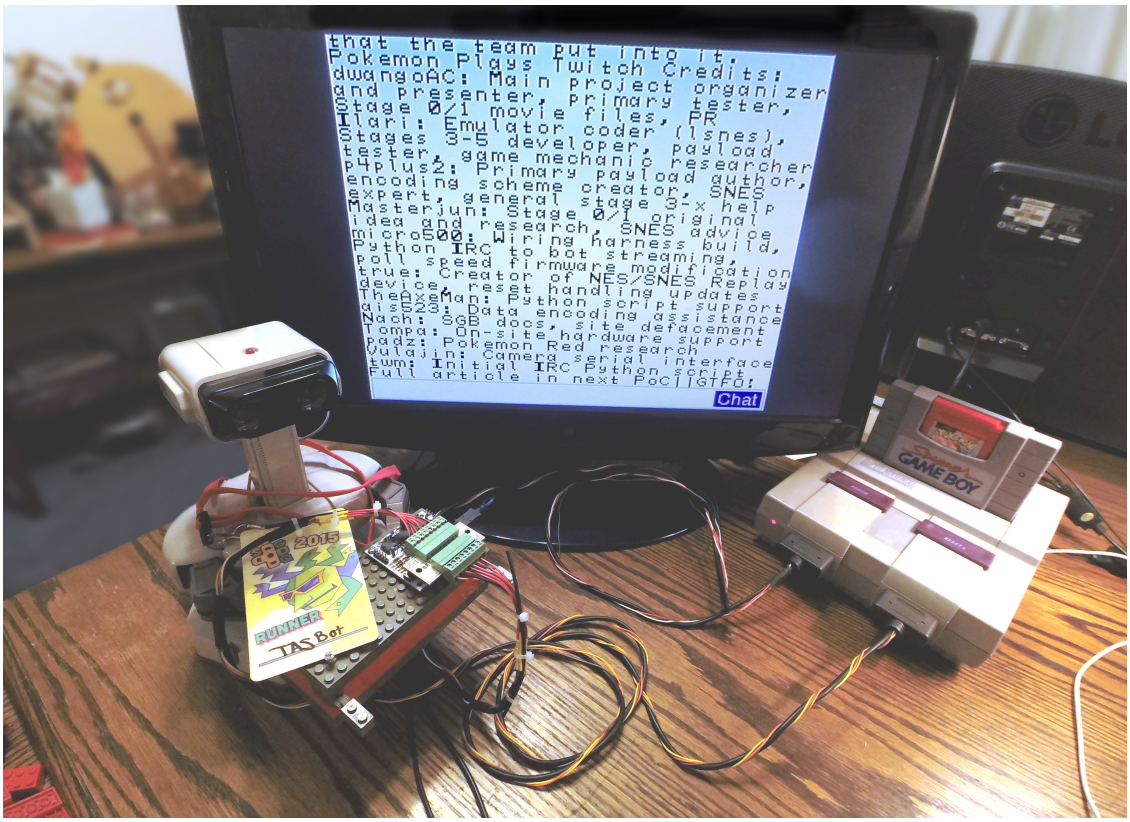

Figure 3 – Exploit with extra credits shown by a real SNES via TASBot

the text box at the bottom of the lsnes Messages window:

```
L start_dump("PPTStage0")
```

Go to Tools → Edit Movie..., right-click in the first column and select Run to frame.... Enter 3457 and hit OK. The movie will play until it reaches the point directly after the reset. Save the dumped movie by executing

```
L stop_dump()
```

which should produce the message "Dumping halted". If everything worked correctly, the resulting file[9] will be created in the working directory of lsnes.

Stage 1 will require creating two separate r16m files which will be recombined along with the Stage 2 movie later. While still on frame 3457, execute

```
L start_dump("PPTStage1P1")
```

and use the same method as before to run up to frame 11992 and stop the dump.

```
L stop_dump()
```

We have to stop here because at this point we need to remove 9 frames of empty input to correct for inaccuracies with the bsnes core's emulation of the clock skew/slip. Use the same Run to frame... method to advance 9 frames to frame 12001. From here we can dump the second portion of Stage 1:

```
L start_dump("PPTStage1P2")
```

and run through to frame 12116 then

```
L stop_dump()
```

The movie will pause at the end of the Stage 1 payload execution, right after Stage 2 is written one nibble per frame.

The next portion needs to switch to a script that will record one latch per poll which will allow the datarate to increase beyond one controller read per frame in later stages. To switch scripts, go to Tools → Reset Lua VM to clear out the frames version then Tools → Run Lua script... → `lsnes_dump_latch_subf.lua`.

As before, type

```
L start_dump("PPTStage2")
```

Run to frame 13273, then

```
L stop_dump()
```

One could stop slightly before this point but if we stop too quickly after the payload is written we will see a repeating buffer of garbled text as was the case during the first AGDQ 2015 presentation; the extra tail here ensures the hardware buffer in the replay board has a consistent stream of empty input.

Concatenate the two portions of Stage 1 and Stage 2 into a single file[10]:

```
cat PPTStage1P1.frame.r16m PPTStage1P2.frame
    .r16m PPTStage2.latsf.r16m >
    PPTStage1and2.r16m
```

We now have the entire payload up to the point where the Twitch chat interface is visible, albeit with no text in the chat area.

It's now possible to take this state and start feeding the article or get some IRC chat text in live.

But feeding directly the recorded IRC chat as available in the polyglot would result in some duplicate input when played on a real console because it was made to handle lag frames in a way that it looks correct on the emulator but which when re-dumped and played on a real console the same characters are sent more than once.

To avoid this, run forward to frame 27780 without dumping a movie.

To extract the payload:

```
L start_dump("PPTStage6")
```

and run to the final frame 29535, then:

```
L stop_dump()
```

To replay r16m files, connect the replay board to a properly populated SNES console with the aforementioned wiring harnesses consisting of all 7 wires for both controller ports. The replay device detection tends to be somewhat finicky when using the serial over USB interface (i.e. connecting the replay board to a computer with a USB cable rather than using the raw serial pins in the corner of the board), so ignore any "Replay device not found" or similar messages you may see in the steps below.

We'll use the official `replay.py`[11] and a modified copy `replay_reset.py` with the following addition:

[9] SHA-256: DBACF452D63F833E7E93148C5421DA4BC95B4386157CD4B5CA5E31C1B919CB38

[10] SHA-256: 0A455C67B29A9ACBA66F78ACDD18840A3EFFDFA6E1C74BA03EC4FB3ACFE4359A

[11] `unzip -j pocorgtfo10.pdf pokemon_plays_twitch/replay*.py`

```
@@ -118,2 +118,3 @@
 ser.write("~v")          # poll for version
+ser.write("~W")          # reset ISCP flag
 time.sleep(0.03)        # usb fix
```

The first replay run will look like this:

```
python replay_reset.py PPTStage0.frame.r16m
    /dev/ttyACM0 0 t 180
```

If everything is connected properly the script will say "Resetting..." and will wait for the console to be turned on. Make sure a copy of Pokemon Red is inserted inside of an SGB cartridge and the Pokemon Red save game has been deleted[12].

If you choose using a reset wire, make sure it is properly connected as described above. Otherwise, to do the reset by hand, practice powering off the console at the correct time by playing through that section of the movie with the emulator a few times to memorize the timing.

Once all conditions are met, turn on the console to start replaying the movie. The rival should then be named `RxRxᴾᵏ`; if things go wrong, menu navigation will be noticeably slower than what you would see on the emulator. At the end of the movie, if the reset wire is present and working the SNES will lock up.

You'll have an intact save file that reports that you have 0 Pokemon *and* you will be able to enter the Pokemon menu and navigate around in empty white space if everything went as expected. From here on out, you won't need the reset pin and any further testing should probably be done without going through this section again.

Because of the pesky timing change between frame mode and latch mode, we can't use the standard `replay.py`. Instead, we'll use the scripts from the PptIrcBot repository, which are also available in the feelies[13] and use the `replay_switch.py` script to run through the stages.

```
python replay_switch.py PPTStage1and2.r16m /
    dev/ttyACM0 0 t 180
```

This should reach the point of an empty Twitch chat interface. You can immediately run the full article text:

```
python replay_stream2.py PPTStage6.latsf.
    r16m /dev/ttyACM0 0 1 180
```

Or you can be even more adventurous and pipe the Internet to your SNES! You can optionally alter `settings.yaml` to your liking, potentially edit `themicrobot.py` and comment out the line `self.ircBot.connection.privmsg(IrcChannel, text)` near line 113 to prevent RED's text from being sent into the actual IRC channel, then run

```
python themicrobot.py
```

in one terminal,

```
python readpipe.py tasbot_pipe
```

in another one (to see what TASBot will be saying), and the following in a third one:

```
touch emptyfile.r16m
python replay_stream2.py emptyfile.r16m /dev
    /ttyACM0 0 1 180
```

If all goes well, your bot user should join the IRC channel you specified and you should hear TASBot speak (assuming you have espeak installed) while the screenplay plays back on your console. More amusingly as demonstrated in the console verification video[14], you can now type whatever you want in the IRC channel and it will be displayed in the SNES Twitch chat interface, even in the middle of replaying the screenplay. While some of the things are clearly a bit contrived (the "web view" shown in the screenplay is nothing more than a carefully palletized screenshot of TASVideos.org, Twitch chat never actually had any influence on the color of the site as shown on the SNES when it was flashing, and we weren't able to finish the camera code in time to make the actual AGDQ 2015 serial-attachable camera function), it's still an impressive feat seeing a game console from 1990 connected to the Internet using only the controller ports.

[12] To show the hidden delete menu, press Select+Up+B when the Title screen is displayed

[13] `unzip -j pocorgtfo10.zip pokemon_plays_twitch/PptIrcBot.zip && unzip PptIrcBot.zip`

[14] `https://youtu.be/NTzrbhCTEhw`