Towards a better Duckyscript toolchain

Vasilij Schneidermann

April 2021

Outline

Section 1

Intro

About

- Vasilij Schneidermann, 28
- Cyber security consultant at msg systems
- mail@vasilij.de
- https://depp.brause.cc
- https://emacsninja.com/

Legal disclaimer

- The contents of this talk are of purely educational nature
- I do not condone usage of any technology presented to hack other people's computers unless with their explicit permission
- Germany has the so-called "Hackerparagraph" which intends to punish people who write software enabling a hacking attempt, but can be interpreted as disincentive to publish hacking tools
- I do not believe this tool to enable attacks that wouldn't have been possible previously, as it merely improves the tooling landscape for those crazy enough to use CHICKEN Scheme, but still...

Previously, at SecCamp 2019

- I've attended an infosec event
- Everyone was handed out a free Digistump Digispark
- I visited the corresponding workshop
- We learned how to weaponize the microcontroller into a USB device sending keystrokes to control the computer
- It was fun and I bought an extra device, for further experimentation

Motivation

- The workshop was cool, but most time was spent messing around with toolchains until we got something running
- Recurring theme at workshops
- I had to use three tools in succession to arm the device with a new payload:
  - Official encoder from Duckyscript to binary (Java)
  - Inofficial translator from binary to Arduino sketch
  - Arduino IDE to compile and deploy code
- Can it be done better?
- Can it support keyboard customizations?
- Can it help you with debugging?

Motivation

- Pranks
- Security assessment (USB stick in the car lot)
- Actual hacking (insert prepared USB stick into unattended computers, collect information, reconfigure machine)

In the news

Secret Service agent Samuel Ivanovich, who interviewed Zhang on the day of her arrest, testified at the hearing. He stated that when another agent put Zhang's thumb drive into his computer, it immediately began to install files, a "very out-of-the-ordinary" event that he had never seen happen before during this kind of analysis. The agent had to immediately stop the analysis to halt any further corruption of his computer, Ivanovich testified. The analysis is ongoing but still inconclusive, he said.

https://www.miamiherald.com/news/politics-government/article228963409.html

Demo

```
DEFAULT_DELAY 500
REM open terminal
WINDOWS ENTER
STRING cat > /dev/null
ENTER
DEFAULT_DELAY 1500
STRING_DELAY 30 Wake up, Neo...
ENTER
STRING_DELAY 50 The Matrix has you...
ENTER
STRING_DELAY 30 Follow the white rabbit.
ENTER
STRING Knock, knock, Neo.
ENTER
CONTROL d
CONTROL d
```

Demo

```
$ plucky -i matrix.duck -o keyboard.c
$ make
$ make deploy
```

Section 2

State of Duckyscript toolchains

Official toolchain

- Available at https://github.com/hak5darren/USB-Rubber-Ducky
- No license (who needs them anyway)
- Designed for original hardware only
- Many third-party payloads:
  - https://github.com/hak5darren/USB-Rubber-Ducky/wiki/Payloads
  - https://ducktoolkit.com/userscripts

Inofficial toolchains

- https://github.com/mame82/duck2spark
- No license (sense a theme?)
- Workflow (taken from README):
  - `echo "STRING Hello World" > test.duck`
  - `java -jar encoder.jar -i test.duck -o raw.bin -l de`
  - `./duck2spark.py -i raw.bin -l 1 -f 2000 -o sketch.ino`
  - Replace sketch contents with generated script
  - Compile and deploy
  - Try out payload

Improvement chances

- One tool, not three
- Fast iteration cycles
- Keyboard customization
- Debugging capabilities
- Supporting both original and custom hardware
- Yet: Behavior parity

Section 3

The BASIC rundown

USB in general

- USB does tons of stuff, tricky to make sense of
- We want this: https://en.wikipedia.org/wiki/USB_human_interface_device_class
- USB HID supports keyboards, mice, game controllers
- There's a standardized list of 256 USB HID codes
- Orthogonal: Mapping to keyboard hardware, OS customizations
- A device sends out USB HID codes with optional modifiers representing key events
- Bonus: The device doesn't need to be a keyboard to do that
- Can be legitimate (barcode scanner) and illegitimate (Rubberducky)

Keyboards in general

- USB HID codes try to be a superset of all possible keys
- Keyboard layout is mapped to these USB HID codes
- Sometimes the mapping is less obvious (ä on QWERTZ maps to `KEY_APOSTROPHE`)
- The OS can implement remapping (swapping modifiers, dead keys, alternative layout key)
- Modifiers can be both modifiers (`KEY_MOD_LCTRL`) and standalone keys (`KEY_LEFTCTRL`)
- Fun fact: `KEY_MEDIA_COFFEE` is a thing

Duckyscript hardware

- USB stick
- Storage to save exfiltrated data to
- Flashed with logic to execute compiled payload
- Price point: 50€

Duckyscript logic

- Program is represented as a stream of bytes
- Every two bytes are either a (modifier) key press or delay
- Key press: Modifier byte and key byte
- Delay: Delay length and zero byte

Digistump Digispark hardware

- General-purpose microcontroller
- Bare board with USB plug
- Not quite USB stick form factor
- Attiny 85 (AVR)
- Little RAM, ROM, no storage
- Price point: 3€

Microcontroller programming

- Harvard architecture: Code separate from data
- Hello world: Making a LED blink
- Your code is not expected to exit
- No OS, no dynamic linking, few abstractions
- Simple solutions are key
- No USB support: Bitbanging

Microcontroller challenges

- Arduino code turns out to be C++
- Refactoring the C++ code into standalone C
- Updating the usbdrv/vusb library
- Initialization
- Waiting for time to pass
- Flash memory access (C is far from ideal for Harvard architectures)
- Figuring out how keys are encoded
- Size optimization

Section 4

Compiling Duckyscript

Duckyscript language

- `REM comment`
- `[MODIFIER] KEY`
- `STRING hello`
- `STRING_DELAY 20 hello again`
- `DELAY 100`
- `REPEAT 3`
- `DEFAULTDELAY 100`

Implementation strategy

- Tokenize
- Parse with a stateful loop
- Generate key press/delay commands
- Verify for errors
- Translate into binary/C code
- Testing

Tokenization and parsing

- I used comparse initially, but it turned out to be overkill for a poorly specified language
- Irregex far better for this, but it's easy to mess up
- Big complicated regexp per line: Slow, hard to understand
- Dispatching on first space-separated token: Fast

Command generation

- Keep track of default delay and last command
- Insert default delay after `STRING`, `STRING_DELAY` and key commands
- Insert last command repeatedly for `REPEAT` command
- Break down `STRING` and `STRING_DELAY` into repeated key presses (with delay if requested)
- Break down key combinations into modifiers and keys
- Break down delays if needed (delay can be 255ms maximum)

Code generation

- I initially tried using `fmt-c`, but it turned out to be overkill
- Now: Template string, with `@identifier@` placeholders
- No template logic though, hardcoded replacements
- Keys are translated to preprocessor macro names

Compiler customization

- Hard to turn C code into binary format
- Sometimes the compiler behaves different from the official one
- Multi-pass approach necessary: Dumping output after each stage
- At the last stage: Decision whether to emit binary or C code

Naming woes

- Everything in the code was called a key
- I let the project sit for almost two years
- Wrote a design document clarifying terms
- Refactored to these names, then adjusted logic
- It works now, tests are broken though...

Test corpus prep

- This slide left blank for legal reasons

Unit testing

```
(when (not (get-environment-variable "TEST_MODE"))
  (main))
```

Unit testing

```
(set-environment-variable! "TEST_MODE" "1")
(include "../plucky.scm")

(test-group ...)

(test-exit)
```

E2E testing

- Wrote helper scripts to convert corpus to binary format with official encoder
- Wrote another test script to convert corpus to binary format with my compiler
- Test script compares the outputs and fails if a difference has been found
- Similar approach could be taken for decompilation

Section 5

Outro

What next?

- Fixing tests
- Decompilation
- Behavior parity
- Actually using it

Questions?