

6 Dumping Firmware from Tamagotchi Friends by Power Glitching

*by Natalie Silvanovich, Tamagotchi Merchant of Death
with the kindest of thanks to Mr. Blinky.*



Figure 5: These sprites were among many dumped from the Tamagotchi Friends ROM.

The Tamagotchi Friends is the latest addition to the Tamagotchi series of virtual pet toys. Released on Boxing Day of 2013, it features NFC messaging and games as a part of a traditional Tamagotchi toy. Recently, I used glitching to dump the code of the Tamagotchi Friends.

The code for the Tamagotchi Friends is stored in mask ROM internal to its GeneralPlus GPLB series LCD controller. In the previous Tamagotchi version (the Tamatown Tama-Go), I used a vulnerability in the processing of external data from a flash accessory to dump the code, but this is not possible for the Tamagotchi Friends, as it does not support flash accessories. In fact, the Tamagotchi Friends has a substantially reduced attack surface compared to the Tamatown Tama-Go, as it also does not support infrared communications. The only available inputs on the Tamagotchi Friends are the buttons, the EEPROM (which is used to store important persistent data, like the number of slices of carrot cake your Tamagotchi has on hand) and NFC.

After eavesdropping on and simulating the NFC, and dumping and rewriting the EEPROM, I determined that they both had limited potential to contain exploitable bugs. They did both appear to fill buffers in RAM with user-controlled data in the course of normal operation though, which meant they both could be useful for creating shellcode buffers in the case that there was a bug that allowed the program counter to be moved to the buffer.

One possible way to move the program counter was glitching, basically driving unexpected signals into the microcontroller and hoping that they would somehow cause that program counter to change and by chance land in the shell code buffer. Considering that memory space of the microcontroller is 65,536 bytes, and the largest buffer I could fill with a NOP slide is roughly 60 non-contiguous bytes this sounds like a long shot, but the 6502 architecture used by the microcontroller has some properties that makes random program counter corruption more likely to lead to code execution compared to other architectures. To start, it has no memory validation, so any access of any address will succeed, regardless of whether any memory is mapped to the location. This means that execution will not stop even if an invalid address is accessed. Also, invalid

opcodes on 6502 are guaranteed to execute in a finite amount of time⁸ with undefined behaviour, so they also will not stop execution. Together, these properties make it very unlikely that execution will ever stop on a 6502 processor, giving shellcode a lot of chances to get executed in the case that the program counter is corrupted.

Another useful feature of this particular microcontroller is that the RAM starts at address zero, and the lowest hundred bytes or so of RAM is used by the SPU and is often zero. In 6502, zero is the opcode for BRK, which acts like NOP if a debugger is not attached, so this RAM could potentially act as a NOP slide. In addition, in the Tamatown Tama-Go (and I assumed the Tamagotchi Friends), the EEPROM is copied to address 0x300, which is still fairly low in RAM addresses. So if the program counter got set to zero, there is a possibility it could slide through RAM up to the EEPROM. Of course, not every value in RAM before 0x300 is zero, but if enough are, it is likely that the other values will be interpreted as instructions that don't alter the program counter's course some portion of the time.

Since setting the program counter to zero seemed especially likely to cause code execution, I started by glitching the input power, as this had the potential to clear the program counter. The Tamagotchi Friends has three types of volatile memory: registers like the program counter, DPRAM (used for the LCD) and SRAM. DPRAM and SRAM both have fairly long persistence after they stop being powered, so I hoped if I cut the power to the microcontroller for a short period of time, it would corrupt the registers, but not the RAM, and resume execution with the program counter at address zero.

I tried this using an Arduino to switch the power on and off at different speeds. For very fast speeds, the Tamagotchi didn't react at all, and for very slow speeds, it would reset every cycle. I eventually settled on cycling every five milliseconds, which had a visible erratic impact on the Tamagotchi after each cycle. At this rate, the toy was displaying an unexpected image on the LCD, corrupting the LCD, playing Yankee Doodle or screeching loudly.

I filled up the EEPROM with a large NOP slide and some code that caused a write to the LCD screen, reset the Tamagotchi so the EEPROM was downloaded into RAM, and cycled the power. Roughly one out of every ten times, the code executed and wrote the LCD.

I then moved the code around to figure out the size of the available code buffer. Two things limited the size. One is that only a small part of the EEPROM is copied into RAM at once, and the rest is only loaded if needed. The second is that some EEPROM addresses are validated. For some of these addresses, containing very critical values, the EEPROM is wiped immediately if the Tamagotchi detects an invalid value. These addresses couldn't be used for code at all. Some other less critical values get overwritten if they are invalid. For example, if a Tamagotchi is a child, but is married, the "is married" flag will be reset to the correct value. These addresses could be changed, but there was no guarantee they would stay the correct value, so I ended up jumping over them. This left exactly 54 bytes for code. It was tight, but I was able to write code that dumped the ROM over SPI through the Tamagotchi buttons in that space

The following is the shellcode I used:

```
SEI ; disable the low battery interrupt
LDA #$FF
```

⁸A few people have mentioned to me that there are certain versions 6502 processors for which this is not true, but this is definitely the case for GeneralPlus controllers.

Protect Your Copies of **BYTE**

NOW AVAILABLE: Custom-designed library files or binders in elegant blue simulated leather stamped in gold leaf.

<p>Binders—Holds 6 issues, opens flat for easy reading. \$9.95 each, two for \$18.95, or four for \$35.95.</p> 	<p>Files—Holds 6 issues. \$7.95 each, two for \$14.95, or four for \$27.95.</p> 
---	--

Order Now!

Mail to: Jesse Jones Industries, Dept. BY, 499 East Erie Ave., Philadelphia, PA 19124

CALL TOLL FREE (24 hours): 1-800-972-8888


Please send _____ files: _____ Name: _____
binders for BYTE magazine. Address: _____ (Use the Office Box)

Enclosed is \$_____ City: _____
Add \$1 per file/binder for postage and handling. Outside U.S. add \$3.50 per file/binder (U.S. funds only please). State: _____ Zip: _____
Charge my: (minimum \$15) _____
 American Express
 Visa MasterCard
 Discover Club

Card # _____
Exp. Date _____
Signature _____

BYTE 

Introducing the Smallest 80386 based PC Compatible Single Board Computer Only 4" x 6"



Quark/PC® II

- VGA® VidedColor LCD Controller
- SCSI Hard Disk Control
- Up to 4 Mbytes Memory and more

To order or enquire call us today.
Megatel Computer Corporation
 (416) 745-7214 FAX (416) 745-8792
 174 Turbine Drive, Weston, Ontario M9L 2S2

Distributors

Germany — Tech Team (06074) 98031 FAX (06074) 90248
 Italy & Southern Europe — NCSItalia (0331) 256-524 FAX (0331) 256-018
 U.K. — Denstron (0959) 76331 FAX (0959) 71017
 Australia — App Microcomputers (03) 500-0628 FAX (03) 500-9461
 Denmark — Ingeniormælet (02) 440-488 FAX (02) 440-745
 Finland — Digipoint (3580) 757 1711 FAX (3580) 757 0844
 Norway — AD Elektronisk (07) 877110 FAX (07) 875990
 Sweden — (040) 9710 90 FAX (040) 43 90 38

Quark is a registered U.S. trademark of Intel Corp. VGA is a registered trademark of IBM Corp.

megatel

```

STA $3011 ; port direction
STA $1109 ; LCD indicator
STA $00C5
STA $00C6
LDX #$08
LDA ($C5),Y ; No room to initialize Y. Worst case,
ASL A ; it will be set to 0 at the end of the loop.
LDY #$01
BCC $001A
LDY #$03
BNE $0020 ; These 4 bytes get altered before execution. Jump over them.
NOP
NOP
NOP
NOP
NOP
STY $3012
LDY #$00
STY $3012
DEX
BNE $0013
INC $00C5
BNE $000F
INC $00C6
BNE $000F
LDA #$00
STA $3000
BNE $000F ; Branches are shorter than jumps, so use implied conditions.

```

In memory, this shellcode is as follows:

```

300: 32 17 02 01 02 01 09 00 1A 00 1A 1A 1A 1A 1A
310: 20 FF 06 10 01 FF FF 02 77 77 77 77 77 77 77
320: 77 77 77 77 77 05 04 FF 77 77 55 00 77 77 7F 00
330: FF FF 40 EA EA EA EA EA 00 00 00 00 00 00 00
340: 03 78 A9 FF 8D 11 30 8D 09 11 8D C5 00 8D C6 00
350: A2 08 B1 C5 0A A0 01 90 02 A0 03 D0 04 EA 00 00
360: 03 EA 8C 12 30 A0 00 8C 12 30 CA D0 E7 EE C5 00
370: D0 DE EE C6 00 D0 D9 4C 4B 03 15 11 4C 38 00 00

```

The code begins at 341 and ends at 376, which are the bounds of the buffer copied from the EEPROM. The surrounding values are typical values of the surrounding RAM which are not consistent across each time code is executed. The 0x03 before the beginning of the code is written after the buffer, and is an undefined instruction in 6502. Unfortunately, this means that there isn't room for any NOP sled, the program counter needs to end up at exactly the right address.

One useful feature of this shellcode is that the first seven instructions aren't strictly necessary! The registers are often the right value, or an acceptable value by chance, which gives the program counter a bit more leeway in the case that it jumps a bit beyond the beginning of the code.

I dumped all thirty-two pages of ROM using this shellcode, and they appear to be accurate. Figure 5 shows the highlights of the dump, organized by cuteness in descending order.