Even More Tamagotchi's Were Harmed in the Making of this Presentation

Natalie Silvanovich
@natashenka
About Me

• Security Researcher at BlackBerry
  – (But I don’t represent them)
• Studied electrical engineering, but mostly into software hacking
• First-time hardware hacker/reverse engineer
• Tamagotchi enthusiast
What are Tamagotchis?

• The same virtual pet toys you remember from the 90’s

• Functionality has evolved substantially
  – Now they can go to school, have jobs, make friends, get married and have kids!

• Newer versions have an IR interface so that they can communicate with other Tamagotchis
TamaTown Tama-Go

- The “Christmas” Tamagotchi from 2010
- Same functionality for smaller hands
- Supports detachable ‘figures’ with extra games and stores
Goals

• Dump Tamagotchi code
• Answer the ‘deeper questions’ of Tamagotchi life
• Make my gotchis rich and happy
• Make a Tamagotchi development environment
• Have fun!
Previous Work
Teardown
Hardware Teardown

• Took apart a Tama-Go and Tamagotchi to determine if code dumping was a possibility
• Looked for helpful interfaces
• Also took apart a figure
Tama-Go Board

EEPROM
Tama-Go Figure
Microcontroller Identification
Identifying the Microcontroller

• Considering the lack of external hardware, MCU was likely under the ‘blob’
• Tried several methods to remove, including acetone, heat, a razor blade and a chopstick
• Travis Goodspeed kindly offered to decap the chip with acid
• Eventually, success!
GPLB5X Series LCD Controller

• 8 bit 6502 microprocessor
• 1536 bytes RAM
• 320 or 640 kbyte mask ROM (depending on model), baked to perfection for each customer
• 512 bytes LCD RAM
• 4 color grayscale LCD controller
• SPI
• Audio DAC
Dumping Mask ROM

• Not sure how to dump mask ROM, but had a few ideas
  – Restore a bad state from EEPROM
  – Look for test functionality
  – Exploit a vulnerability in figure or IR processing
  – Read ROM with a microscope
  – Pin manipulation
Test Program
Test Program?

• GeneralPlus mask ROMs contain a GP test program that can probably dump code
• Contacted GeneralPlus for a devkit
  – Requires an NDA
• Looked around online
  – No one seems to have a devkit or know the test program
Figure ROM
Figure ROM

• Decoding the figure ROM could be useful in a few ways
  – Making your own Tamagotchi games
  – Executing code on the Tamagotchi
  – Dumping mask ROM
  – Understanding Tamagotchi behaviour
Figure ROM Pads

- The unpopulated PCBs in lite figures appear to be the same boards used in regular figures.

- Makes the mask ROM pad layout visible.
Figure ROM Chip

- GeneralPlus makes an SPI ROM with a similar layout

- Assumed figures use this ROM
Based on the GeneralPlus ROM datasheet, was able to identify the figure pins:

1, 4 and 8: Ground/Jumper
2: Serial clock (C)
3: Serial data input (D)
5: Power
6: Chip Select (SB)
7: Serial Data Output (Q)
ROM Dump

• Dumped the ROM using an Arduino as SPI master
Decoding ROM

• The Tamagotchi has a four-tone display, so looked for strings of 0x00, 0x55, 0xAA and 0xFF, representing images

• Noticed that these strings were preceded by values which were reasonable for length and width
Decoding Images

• Tried decoding these images

• Eventually, it worked!
Images

- The figure contained a lot of images
- Text displays appear to be images

- Animations are series of images
The Rest of the ROM

• The ROM contains a lot of non-image data
• None of this data is GeneralPlus code
  – Wrote a disassembler
• Likely logic information in some sort of serialized format
Simulating the ROM

- Could not obtain compatible flash
- Attempted to simulate the ROM using an Arduino, but chip is too slow
- Switched to a Chipkit Uno, this was also too slow
- Eventually used a STM32F4 Discovery board
Simulating the ROM

- Knew the image format, so could alter images
Game Logic

• The Tama-Go reads less than 50 bytes of non-image data during all figure functionality

• Game logic is represented by a one byte code
  – This logic is executed with images from figure

• Changing this code can cause a jump to non-game screens
  – Stats, food, death, etc. Every screen was available

• Many codes caused freezing
Evolve Demo
Flash Figures
Flash Figures

• MrBlinky ordered a set of figures to experiment with
  – They contained flash!
  – Built a figure programmer
  – The ability to re-flash figures made testing much easier
Items

- Items are implemented using a byte code format
  - Instructions include showing images, playing sounds and changing Tamagotchi stats
  - Some unusual behaviour for invalid instructions
  - Posted ‘dev tools’ on github
Demo

```
GAME
SHOP
ITEM
```
Code Execution
Game Logic

• The Tama-Go reads less than 50 bytes of non-image data during all figure functionality
• Game logic is represented by a one byte code
  – This logic is executed with images from figure
• Changing this code can cause a jump to non-game screens
  – Stats, food, death, etc. Every screen was available
• Many codes caused freezing
6502 Facts

- Memory mapped into a single address space
- No MMU
  - Unmapped addresses return 0 (usually)
  - Invalid instructions execute undefined behaviour
- Reset is rare
  - Great for exploitation
First Attempt

• Assumed ‘game codes’ were indexes into a jump table
  – Invalid indexes would cause jumps (RTS) to non-pointer data
• Only controllable RAM is LCD RAM
  – 0x1000-0x1200
• Made a NOP sled and hoped
Code 0xCC

• Did not work, but code 0xCC had interesting behavior
  – Buzzed when bit 3 of byte 68 was set and detected figure detach
  – Froze otherwise
• Also noticed that some middle indexes worked
New Theory

• All indexes are valid, but the stack isn’t set up correctly
• 0xCCCC plays the noise when button pressed

```python
if sound_enabled:
    play_sound()
    jump to a
else:
    jump to b
```

LCD RAM

Game code jump table address

???
New Theory

• But if
  – A pointer to the LCD RAM is on the stack
  – Stack confusion is occurring
  – There’s 255 possibilities
• Why isn’t it working?
<table>
<thead>
<tr>
<th>NO</th>
<th>SYNTAX 6502</th>
<th>SYNTAX 2500</th>
<th>6502 SUN bc type</th>
<th>addressing modes</th>
</tr>
</thead>
<tbody>
<tr>
<td>001</td>
<td>ADC #dd</td>
<td>ADC A,dd</td>
<td>69H 56h 2 2 cpu3</td>
<td>immediate</td>
</tr>
<tr>
<td>002</td>
<td>ADC aa</td>
<td>ADC A,aa</td>
<td>65H 17h 2 2 cpu3</td>
<td>zero page</td>
</tr>
<tr>
<td>003</td>
<td>AND #dd</td>
<td>AND A,dd</td>
<td>29H 54h 2 2 cpu3</td>
<td>immediate</td>
</tr>
<tr>
<td>004</td>
<td>AND aa</td>
<td>AND A,aa</td>
<td>25H 15h 2 2 cpu3</td>
<td>zero page</td>
</tr>
<tr>
<td>005</td>
<td>BCC ??</td>
<td>JR NC,??</td>
<td>98H 28h 2 2 cpu3</td>
<td>relative</td>
</tr>
<tr>
<td>006</td>
<td>BCS ??</td>
<td>JR C,??</td>
<td>B8H 28h 2 2 cpu3</td>
<td>relative</td>
</tr>
<tr>
<td>007</td>
<td>BEQ ??</td>
<td>JR Z,??</td>
<td>F0H 3Ah 2 2 cpu3</td>
<td>relative</td>
</tr>
<tr>
<td>008</td>
<td>BIT aa</td>
<td>BIT &lt;aa&gt;</td>
<td>24H 11h 2 2 cpu5</td>
<td>zero page</td>
</tr>
<tr>
<td>009</td>
<td>BIT aaaa</td>
<td>BIT &lt;aaaa&gt;</td>
<td>2CH 51H 3 4 cpu5</td>
<td>absolute</td>
</tr>
<tr>
<td>010</td>
<td>BMI ??</td>
<td>JR M,??</td>
<td>30H 18h 2 2 cpu3</td>
<td>relative</td>
</tr>
<tr>
<td>011</td>
<td>BNE ??</td>
<td>JR NZ,??</td>
<td>D8H 2Ah 2 2 cpu3</td>
<td>relative</td>
</tr>
<tr>
<td>012</td>
<td>BPL ??</td>
<td>JR P,??</td>
<td>18H 08h 2 2 cpu3</td>
<td>relative</td>
</tr>
<tr>
<td>013</td>
<td>BRK</td>
<td>BRK</td>
<td>00H 00H 1 7 cpu3</td>
<td>implied</td>
</tr>
<tr>
<td>014</td>
<td>BUS ??</td>
<td>JR NOV,??</td>
<td>58H 0Ah 2 2 cpu3</td>
<td>relative</td>
</tr>
<tr>
<td>015</td>
<td>CALL aa,aa</td>
<td>CALL A,aa</td>
<td>18H 40H 1 4 cpu3</td>
<td>implied</td>
</tr>
<tr>
<td>016</td>
<td>CLC</td>
<td>CLC</td>
<td>58H 4Ah 1 4 cpu3</td>
<td>implied</td>
</tr>
<tr>
<td>017</td>
<td>CLI</td>
<td>CLI</td>
<td>58H 78H 1 4 cpu3</td>
<td>implied</td>
</tr>
<tr>
<td>018</td>
<td>CMP #dd</td>
<td>CP A,dd</td>
<td>9CH 66H 2 2 cpu3</td>
<td>immediate</td>
</tr>
<tr>
<td>019</td>
<td>CMP aa</td>
<td>CP A,aa</td>
<td>C9H 27H 2 2 cpu3</td>
<td>zero page</td>
</tr>
<tr>
<td>020</td>
<td>CMP aa,X</td>
<td>CP A,&lt;aa+X&gt;</td>
<td>D5H 2FH 2 4 cpu3</td>
<td>zero page indexed x</td>
</tr>
<tr>
<td>021</td>
<td>CPX #dd</td>
<td>CPX X,dd</td>
<td>E0H 32H 2 2 cpu3</td>
<td>immediate</td>
</tr>
<tr>
<td>022</td>
<td>CPX aa</td>
<td>CPX X,aa</td>
<td>E4H 33H 2 2 cpu3</td>
<td>zero page</td>
</tr>
<tr>
<td>023</td>
<td>DEC aa</td>
<td>DEC &lt;aa&gt;</td>
<td>C6H A3H 2 5 cpu3</td>
<td>zero page</td>
</tr>
<tr>
<td>024</td>
<td>DEC aa,X</td>
<td>DEC &lt;aa+X&gt;</td>
<td>D6H A0H 2 6 cpu5</td>
<td>zero page indexed x</td>
</tr>
<tr>
<td>025</td>
<td>EOR #dd</td>
<td>EOR A,dd</td>
<td>49H 24h 2 2 cpu5</td>
<td>implied</td>
</tr>
<tr>
<td>026</td>
<td>EOR aa</td>
<td>EOR A,aa</td>
<td>55H 0FH 2 4 cpu5</td>
<td>zero page indexed x</td>
</tr>
<tr>
<td>027</td>
<td>INC aa</td>
<td>INC &lt;aa&gt;</td>
<td>E6H B3H 2 5 cpu3</td>
<td>zero page</td>
</tr>
<tr>
<td>028</td>
<td>INX</td>
<td>INC X</td>
<td>E8H 72H 1 2 cpu3</td>
<td>implied</td>
</tr>
<tr>
<td>029</td>
<td>JMP aaaa</td>
<td>JP aaaa</td>
<td>4CH 43H 3 3 cpu3</td>
<td>absolute</td>
</tr>
<tr>
<td>030</td>
<td>JSR aaaa</td>
<td>CALL aaaa</td>
<td>28H 10H 3 6 cpu3</td>
<td>absolute</td>
</tr>
<tr>
<td>031</td>
<td>JX</td>
<td>JX</td>
<td>22H 77H 2 1 cpu3</td>
<td>immediate</td>
</tr>
<tr>
<td>032</td>
<td>LDA #dd</td>
<td>LDA A,dd</td>
<td>99H 74H 2 2 cpu3</td>
<td>immediate</td>
</tr>
<tr>
<td>033</td>
<td>LDA aa</td>
<td>LDA A,aa</td>
<td>A5H 35H 2 3 cpu3</td>
<td>zero page</td>
</tr>
<tr>
<td>034</td>
<td>LDA aa,X</td>
<td>LDA A,&lt;aa+X&gt;</td>
<td>B8H 45H 2 4 cpu3</td>
<td>zero page indexed x</td>
</tr>
<tr>
<td>035</td>
<td>LDAA aa,aa</td>
<td>LD A,&lt;aaaa&gt;</td>
<td>A9H 75H 3 4 cpu3</td>
<td>absolute</td>
</tr>
<tr>
<td>036</td>
<td>LDAA aaaa</td>
<td>LD A,&lt;aaaa&gt;</td>
<td>BDH 7DH 3 4 cpu3</td>
<td>absolute indexed x</td>
</tr>
<tr>
<td>037</td>
<td>LDX #dd</td>
<td>LDX X,dd</td>
<td>A2H 0B0H 2 2 cpu3</td>
<td>implied</td>
</tr>
<tr>
<td>038</td>
<td>LDX aa</td>
<td>LDX X,aa</td>
<td>A6H 81H 2 3 cpu3</td>
<td>zero page</td>
</tr>
<tr>
<td>039</td>
<td>NOP</td>
<td>NOP</td>
<td>EEH F1H 3 4 cpu5</td>
<td>absolute</td>
</tr>
<tr>
<td>040</td>
<td>ORA #dd</td>
<td>OR A,dd</td>
<td>09H 44H 2 2 cpu3</td>
<td>immediate</td>
</tr>
<tr>
<td>041</td>
<td>ORA aa</td>
<td>OR A,aa</td>
<td>B9H 05H 2 3 cpu3</td>
<td>zero page</td>
</tr>
<tr>
<td>042</td>
<td>PHP</td>
<td>PHP</td>
<td>00H 40H 1 4 cpu3</td>
<td>implied</td>
</tr>
<tr>
<td>043</td>
<td>PLA</td>
<td>PLA</td>
<td>68H 52H 1 4 cpu3</td>
<td>implied</td>
</tr>
<tr>
<td>044</td>
<td>PLP</td>
<td>PLP</td>
<td>28H 50H 1 4 cpu3</td>
<td>implied</td>
</tr>
<tr>
<td>045</td>
<td>ROL #dd</td>
<td>ROL A,dd</td>
<td>2AH D0H 1 2 cpu3</td>
<td>accumulator</td>
</tr>
<tr>
<td>046</td>
<td>ROL aa</td>
<td>ROL A,aa</td>
<td>26H 91H 2 5 cpu3</td>
<td>zero page</td>
</tr>
<tr>
<td>047</td>
<td>ROR #dd</td>
<td>ROR A,dd</td>
<td>6AH 2DH 1 2 cpu3</td>
<td>accumulator</td>
</tr>
<tr>
<td>048</td>
<td>ROR aa</td>
<td>ROR A,aa</td>
<td>6AH 2DH 1 2 cpu3</td>
<td>accumulator</td>
</tr>
</tbody>
</table>
Code Execution

• Switched instruction sets
• Used simpler shellcode
• Using the correct instruction set, it worked on the fourth index I tried, 0xd4
Dumping ROM
dumping memory

- Wrote code to dump entire memory space of Tamagotchi
- Output memory over SPI using port A (buttons)
- Decoded output with signal analyzer
Paging

• The ROM is larger than the memory space
• First page is always mapped
• Other pages are mapped one at a time
• Determined 0x3000 is page port
• Dumped all 19 pages
Pages

• Quickly identified pages by inspection
  – Pages 0 to 6 are code
  – Pages 7 to 9 are blank
  – Page 10 contains images and a image pointer table
  – Pages 11 to 18 contain image data
  – Page 19 contains audio
Images

• Dumped images from image pages
ROM Reversing

• Started using IDA
  – Learning curve was steep
  – No paging support
• Eventually wrote a simulator based on py65
  – Added support for LCD and ports
  – Slowly decoded the secrets of Tamagotchi life
Better Emulator

- Asterick wrote a JavaScript-based emulator
  - https://github.com/asterick/tamago
Tamagotchi Internals

• After start-up, Tamagotchis cycle through a single loop, driven by tm1 interrupts

• Always in one of 0x41 states
  – Table determines state actions
  – Can have substates and subsubstates and ...
  – State entry behaves differently
  – States are responsible for all behaviour (buttons, sound) except for physical LCD update and SPI poll
  – A LOT of pointer tables
Secrets So Far ...

• What makes a Tamagotchi a boy or a girl?
  – Determined from entropy source C4, based on how many times tm1 has fired since the Tamagotchi started up

• What toddler a baby grows into is random
  – Intentionally evened out
  – Some toddlers are higher-maintenance than others
Secrets So Far ...

• What teen a toddler becomes is based on care
  – Two factors
• What adult a teen becomes depends on care and training
  – Toddler care matters
• You can potty train your Tamagotchi
Test mode

- Uncovered a test mode if figure ID is 0xFE
Test Mode

- Allows all stats to be altered
- Allows character and spouse to be selected
- Allows care factors to be viewed and altered
- Two unused care factors
More Secrets

• It doesn’t matter who your Tamagotchi marries
  – They’re just as happy
  – The kids turn out just the same
    • Unless you marry an Olditchi

• Figures don’t alter Tamagotchi functionality outside of their functionality
  – Special display for 100 figures
Just be aware user or tamatalk cannot be held responsible if you do these tasks. These are your choice, at your own risk.

Interesting.

*cough* Makiko and Shimashimatchi *cough...*

Interesting, you are putting much effort in something that most consider not worth it, kudos to you 🌟
HATERS GONNA HATE
Test Program
GeneralPlus Test Program

- Analyzed GeneralPlus Test Program
- Hoped it would make dumping other GP ROMs easier
GeneralPlus Test Program

• Polls port A for a code, runs test and outputs results on port B
• Two interesting codes, 3 and 0x16
• Code 3 checksums custom address range
  – Unfortunately contains a bug so it doesn’t work
Test Program Code Dump

• Code 16 fills RAM up with code from Port B and jumps to it!
• Can dump code from any GeneralPlus LCD controller so long as Port A, Port B and TEST are bonded
Dev Tools
Existing Tools

• Wrote two ‘dev’ tools in the process of reversing
  – portrait.py puts an image on the Tamagotchi screen
  – itemmake.py makes a ‘music video’ based on a script

• Both have serious limitations

• Wanted to write a tools that allows generic 6502 execution
Reliable Exploitation
Reliable Exploitation

• The vulnerability used to dump the ROM was 30-40% reliable
  – Worked better if the Tamagotchi had been running awhile
• Needed 100% reliability for a useful dev tool
The ROM Dump Vuln (D4)

- The game indices in the figure ROM cause a state change to 0x27 + the index

```
seg004:4E2E       LDA   byte_1A4
seg004:4E31       BEQ   loc_44E39
seg004:4E33       LDA   gameindex2
seg004:4E36       JMP   loc_44E3C
seg004:4E39       ;  -----------------------------
seg004:4E39       seg004:4E39:    LDA   gameindex1
seg004:4E39       seg004:4E3C    CLC
seg004:4E3C       loc_44E3C:    ADC   #$27 ;...
seg004:4E3C       STA    current_state_22
seg004:4E3D       JMP    locret_44E4C
seg004:4E3F
seg004:4E41
```

- Valid indices are between 0 and 0x41 – No validity check
The ROM Dump Vuln (D4)

• On a state change
  – Tamagotchi indexes into a state page table, switches to the page at the index and jumps to $0x4000$
  – Code pages have code at $0x4000$ that indexes into a jump table for the page
  – Invalid states could cause a jump to a non-code page, or a jump to an unexpected address
The ROM Dump Vuln (D4)

• State is set to 0x27 + 0xD4 (0xFB)
  – Page table returns 0x3c (actually part of LCD table)
• Switching to page 0x3c makes memory at 0x4000 float
  – No wonder this exploit is unreliable
Vulnerability Idol

• Finding a more reliable index required a lot of tracing
• Eventually tried several indexes to find one that seemed reliable
  – 0xCD was a good contender
Index 0xCD

• State is set to 0x27 + 0xCD (0xF4)
  – Page table returns 0x4 (also part of LCD table)
• Loads page 4 and indexes jump table at 0xF4
  – This location is actually code: INC $11E
  – As data, it resolves to location 0x1EEE
  – LCD RAM addressing ignores bits 2-7 of byte 3
  – Resolves to 0x10EE (in LCD RAM)
• This exploit will always work
Dev Kit
tASMgotchi

- 6502 Assembler for Tamagotchi
- Outputs binary ready to be loaded on figure
- Loads code into RAM, and automatically handles paging during execution
- Contains convenience functions for common functionality such as LCD writes and IR
  - Largely from Tamagotchi ROM
- Ophis based
Making the Dev Kit

• Lack of datasheet made writing some functions difficult
  – Limited knowledge of port locations

• Determined a lot of functionality from the test program

• Still a lot of unknowns
  – Power management, SPU, watchdog
  – Contributions welcome!
Making the Dev Kit

- Egg Shell board
- SPI programmer and IR for future RCE 😊
- Also a Lilypad USB Arduino
Tamagotchi Tools

https://github.com/natashenka/Egg-Shell

- Portrait maker
- Item maker
- tASMgotchi
- Board specs
Workshop

Learn to hack Tamagotchis here at 30c3!

Today at 7:30pm in Hall E

Kit is €25 + VAT, and includes a Tamagotchi, figure and a programming board
Egg Shell Boards

• Boards €11, PCBs €2

• http://natashenka.ca/boards/
Demo
Buttons
Conclusion
Conclusions

• Dumped Tamagotchi code
• Learned about Tamagotchi internals
• Learned the secrets of Tamagotchi life
• Made Tamagotchitis do new things
• Most importantly, good times were had by all...
Except for the Tamagotchis
Tamagotchi Friends
A New Tamagotchi!
Tamagotchi Friends

• Similar LCD and form factor
  – No IR or figures
  – Contains NFC
• Send gifts
• Visit
• Send messages
• Daily limits
Is it Hackable?

• Tamagotchi Friends probably uses the same MCU as the Tama-Go
  – Same form factor and LCD

• If it does, code can be dumped using the GeneralPlus test program
  – Decapping may be required
  – Reduced attack surface for code execution

• If not, who knows?
More Info

http://natashenka.ca
natalie@natashenka.ca
@natashenka