



Cyber Security and Video Game Speedrunning



Jordan



Mike



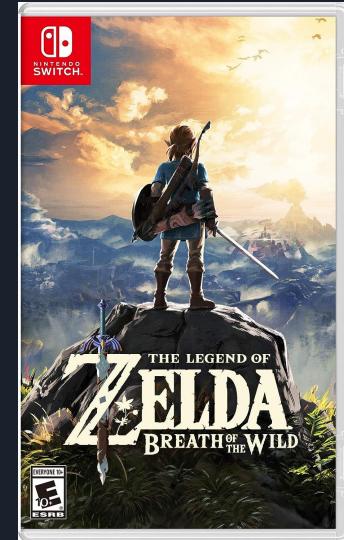


Quick Overview

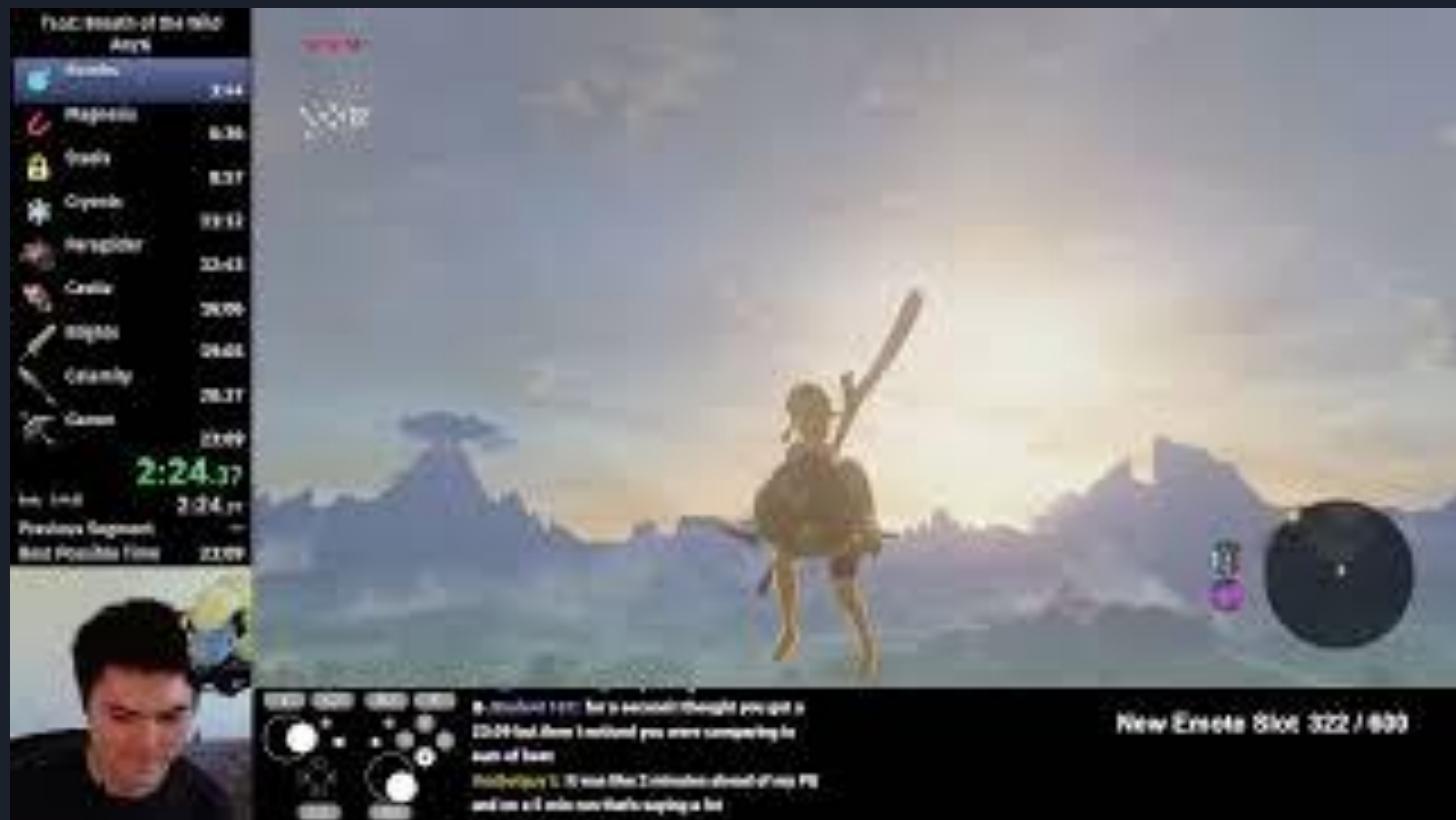
- Basic Speedrunning
- Tool-Assisted Speedrunning (TAS)
- Emulation
- Reverse Engineering
- Fuzzing
- Hardware Hacking
- Arbitrary Code Execution

Speed Running

- Playing a video game as fast as possible
 - Any % - get to the ending as fast as possible
 - 100% - full completion of game (collect all items, play all levels)
 - Glitchless - can't use glitches during gameplay
- How fast can you play through Zelda BotW or TotK?
 - I'm 110 hrs into TotK, only have 3 heroes with me
 - BotW took me over 100 hrs...
- Some records from Speedrun.com
 - Zelda: Breath of the Wild takes < 24 min
 - Zelda: Tears of the Kingdom < 44 min
- 2 Big Speedrunning Marathons each year for charity
 - Jan - Awesome Games Done Quick (AGDQ)
 - Jun - Summer Games Done Quick (SGDQ)

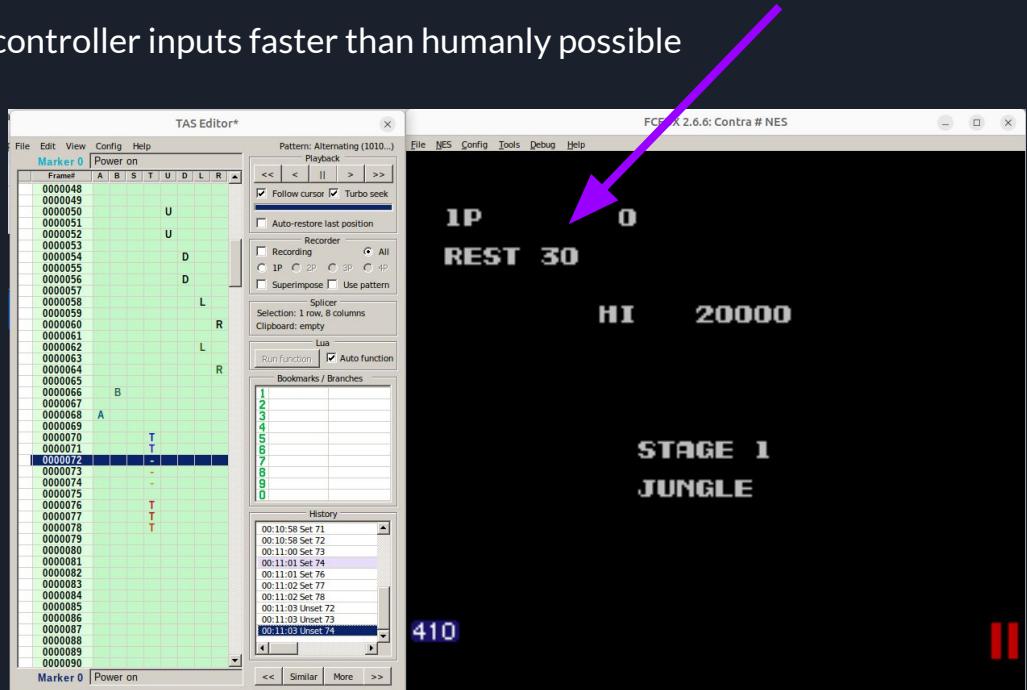


Zelda BotW Speedrun Excerpt



Tool-Assisted Speedrunning (TAS)

- Precise manipulation of all controller inputs faster than humanly possible
- Typically Requires
 - Emulation
 - Save States
 - Frame by Frame Control inputs
- Doom Done Quick (1999)
- Contra - Konami Code



Emulators

- Software that enables a computer system (host) to behave like another (guest)
- Have a way to interact with guest (I/O)
 - Controller inputs
 - Networking
 - Keyboard / Mouse
 - Graphics
- Simulation CPU operation
 - Understand and implement all assembly instructions
 - Processor interrupts / exceptions
- Memory
 - ROM Chips
 - Virtual addressing modes
- Debugging
 - Stepping through code as it executes



bizhawk
a multi-system emulator



Speedrunning vs Security Researcher



Feature	Speed Runner	Security Researcher
Save State	Rewind to improve each segment of speed run	Freeze malware before it starts malicious activity.
Crafting inputs	Precise controls / frame perfect inputs	Crafting the perfect set of bytes to trigger vulnerability or exploit
Memory Introspection via Debugger	How can I manipulate game state / PRNG	Examine memory contents after this virus deobfuscates itself
No original hardware required	Old consoles unreliable, analog video, expensive.	Real hardware may not be available, rare, too expensive.

Reverse Engineering

- What controller inputs make me go fastest?
- What attacks do the most damage?
- Live Split timing - inspects memory to detect run time / changes in game state
- Trainers (cheats)
- Cracking copy protection (not piracy)
 - Floppy disks with fuzzy bits / busted sectors
 - DRM that breaks games on newer operating systems
 - Does your PC still have a CD-ROM or floppy drive?
 - Rockstar games has repeatedly sold cracked copies of their own games on Steam
 - Old hardware that doesn't work

	The Legend of Zelda: The Wind Waker	Any% (English, No Tuner)	60
	Hero's Sword	-4.7	10:50
	Leaving Outset		18:24
	Forsaken Fortress 1		29:23
	Wind Waker		38:59
	Empty Bottle		42:17
	Delivery Bag		47:00
	Kargaroc Key		52:21
	Grappling Hook		55:27
	Enter Gohma		56:38
	Dragon Roost Cavern		59:38
	Northern Triangle		1:02:53
	Greatfish		1:04:01
	Bombs		1:09:33
	Deku Leaf		1:17:13
	Enter Kalle Demos		1:21:33
	Ganondorf		4:51:34
10:56.48			
Previous Segment			-4.7

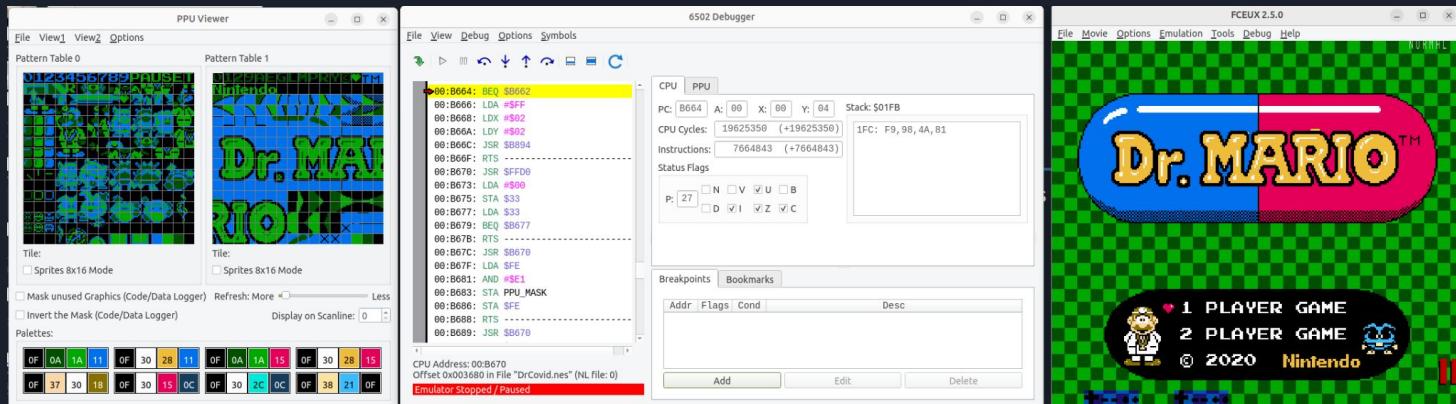


Rom Hacks / Patching

- Randomizer
 - Change item locations so game route has to change randomly
- Level Customizations:
 - Kaizo
- Relay Races
- Regular ROM Hacks
 - Language translations
 - Quality of life
 - Custom levels / Artwork
 - Extra Difficulty
- Disable anti-debugging / security features
- Disable encryption on comms
- Add in secret features

Reverse Engineering Tools

- Convert binary data from ROM into assembly instructions / game data
- Convert assembly code into high-level code / C code
- Create data structures / further understand code
- Assign names to function / variables
- Go back and forth between debugging and static analysis



Reverse Engineering Tools

The screenshot shows the Binary Ninja interface with the file `flash_and_ram.bnbd` open. The assembly view displays the following code:

```
void* first_challenge_screen()

10000e9e    oled_display(string_to_print: "Collect the Measures", x_pos: 4, y_pos: 0xf, inverted_flag: 0)
10000ea0    oled_display(string_to_print: "Break the Silence", x_pos: 0xa, y_pos: 0x1b, inverted_flag: 0)
10000eba    return oled_display(string_to_print: "Continue the Journey", x_pos: 4, y_pos: 0x27, inverted_flag: 0)

10000ecb    int32_t data_10000ebc = 0x1000d584
10000ec0    int32_t data_10000ec0 = 0x1000d59c
10000ec4    int32_t data_10000ec4 = 0x1000d5b8

10000ec8    int32_t challenge_complete()

10000ed8    for (int32_t i = 0; i <= 0xe; i = i + 1)
10000ed2        *(0x20005cc + i) = 0
10000edc    memsetFullFrameBuffer(0)
10000ef4    oled_display(string_to_print: "YOU DID IT!", x_pos: 0x1f, y_pos: 0, inverted_flag: 0)
10000ef4    oled_display(string_to_print: "Check the challenges", x_pos: 4, y_pos: 0xa, inverted_flag: 0)
10000f00    oled_display(string_to_print: "menu for the", x_pos: 0x1c, y_pos: 0x14, inverted_flag: 0)
10000f0c    oled_display(string_to_print: "next step.", x_pos: 0x22, y_pos: 0x1e, inverted_flag: 0)
10000f18    oled_display(string_to_print: "Press DOWN", x_pos: 0x22, y_pos: 0x2d, inverted_flag: 0)
10000f24    oled_display(string_to_print: "to continue.", x_pos: 0x1c, y_pos: 0x37, inverted_flag: 0)
10000f28    int32_t r8 = unknown_i2c_stuff()
10000f30    while ((zxd(data_20026ef6_read_by_lots) << 0x1e) >= 0)
10000f30        nop
10000f38    g_GlobalMenuStateMaybe = 1
10000f3a    return r8

10000f3c    int32_t data_10000f3c = 0x200055cc
10000f40    int32_t data_10000f40 = 0x1000d5c8
10000f44    int32_t data_10000f44 = 0x1000d5d4
10000f48    int32_t data_10000f48 = 0x1000d5ec
10000f4c    int32_t data_10000f4c = 0x1000d5fc
10000f50    int32_t data_10000f50 = 0x1000d608
10000f54    int32_t data_10000f54 = 0x1000d614
10000f58    int32_t data_10000f58 = 0x20026ef6
10000f5c    int32_t data_10000f5c = 0x20026efb

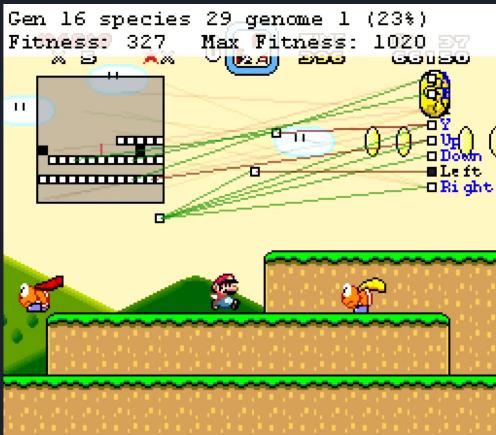
10000f60    void* display_credits()

10000f6a    drawFillRect(xStart: 0, yStart: 0, xStop: 0x7f, yStop: 8)
```

The assembly view includes a sidebar with symbols and cross-references. The status bar at the bottom right shows memory addresses and byte counts.

Fuzzing

- Send random data (or corrupt / randomize parts of good data) into a software system
- Monitor the system and check for anomalous outputs
- Repeat until things break, study the input, determine why the input breaks the system
- TAS: Marl/O, Mario FLOW (Mario Kart AI), PlayFun, Bizqwit TAS runs
- Fuzzers: AFL, AFL++, libFuzzer



```
american fuzzy lop 2.42b (xmllint)
process timing
  run time : 2 days, 7 hrs, 11 min, 16 sec
  last new path : 0 days, 0 hrs, 26 min, 14 sec
  last uniq crash : none seen yet
  last uniq hang : none seen yet
cycle progress
  now processing : 2064 (37.93%)
  paths timed out : 1 (0.02%)
stage progress
  now trying : arith 8/8
  stage execs : 113k/12.7M (0.89%)
  total execs : 20.5M
  exec speed : 33.91/sec (slow!)
fuzzing strategy yields
  bit flips : 1410/3.34M, 120/1.89M, 118/1.89M
  byte flips : 1/235k, 32/235k, 24/234k
  arithmetics : 347/2.98M, 0/193k, 0/0
  known ints : 71/298k, 21/1.48M, 11/2.33M
  dictionary : 0/0, 0/0, 98/2.47M
  havoc : 1050/1.39M, 0/0
  trim : 0.36%/26.5k, 0.12%
overall results
  cycles done : 0
  total paths : 5441
  uniq crashes : 0
  uniq hangs : 0
map coverage
  map density : 7.37% / 18.75%
  count coverage : 4.04 bits/tuple
findings in depth
  favored paths : 851 (15.64%)
  new edges on : 1254 (23.05%)
  total crashes : 0 (0 unique)
  total tmouts : 2921 (269 unique)
path geometry
  levels : 2
  pending : 5229
  pend fav : 733
  own finds : 3310
  imported : n/a
  stability : 98.53%
[cpu000:101%]
```

Hardware Hacking

- Build custom circuits to get controller inputs
Into the console
- Electrical Engineering skills
 - Digital circuit design
 - Custom FPGAs

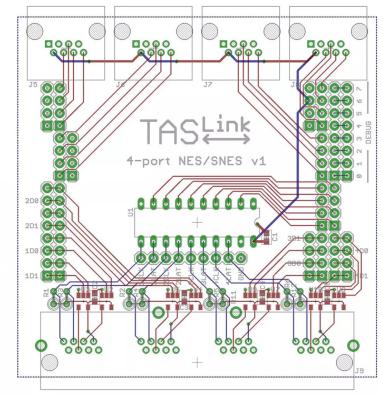
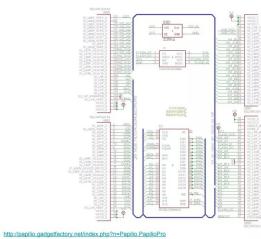


Hacker Badges!

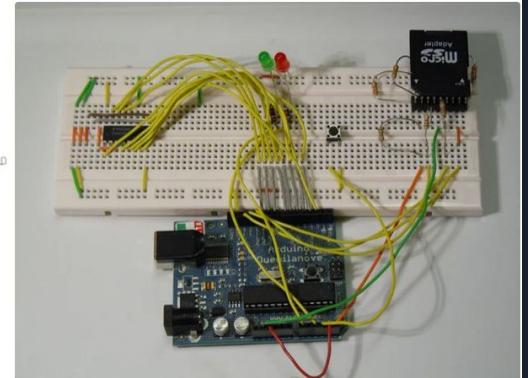
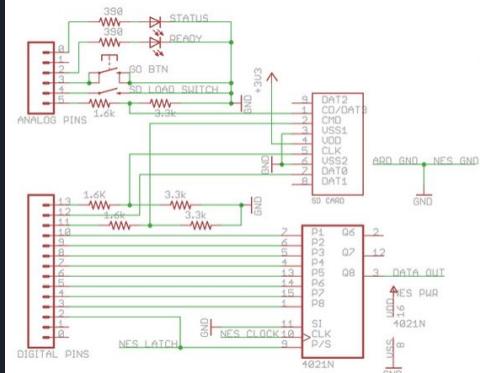
32Mhz FPGA

Papilio Pro's Spartan 6 LX

max poll rate of
the serial port (2Mb/s)



Micro500 - Early TAS hardware, ran script from SD card



Arbitrary Code Execution

- Using item manipulations / memory corruption bugs
 - Change the flow of code of a program
- Credits warp
- Adding whole new games / modes / code
 - Snake/Pong game in Super Mario World
 - Pokemon Red with Twitch Chat on SNES
 - Pokemon Yellow with Tetris, SMB, Portal music



```
$ arm-none-eabi-gdb -q ./nes/mii/mtftp
Using host libthread_db library "/lib/tls/libc-2.14.so/cnvv/libthread_db.so.1".
Using target libthread_db library "/lib/tls/libc-2.14.so/cnvv/libthread_db.so.1".
(gdb) target remote localhost:12345
(gdb) break main
Breakpoint 1 at 0x404144: file main.c, line 19.
Starting program: ./nes/mii/mtftp -q (core dumped)
(gdb) p $pc
$1 = 0x404144 <main+7>
(gdb) p $r0
$2 = 0x404144 <main+7>
(gdb) p $r1
$3 = 0x404144 <main+7>
(gdb) p $r2
$4 = 0x404144 <main+7>
(gdb) p $r3
$5 = 0x404144 <main+7>
(gdb) p $r4
$6 = 0x404144 <main+7>
(gdb) p $r5
$7 = 0x404144 <main+7>
(gdb) p $r6
$8 = 0x404144 <main+7>
(gdb) p $r7
$9 = 0x404144 <main+7>
(gdb) p $r8
$10 = 0x404144 <main+7>
(gdb) p $r9
$11 = 0x404144 <main+7>
(gdb) p $r10
$12 = 0x404144 <main+7>
(gdb) p $r11
$13 = 0x404144 <main+7>
(gdb) p $r12
$14 = 0x404144 <main+7>
(gdb) p $r13
$15 = 0x404144 <main+7>
(gdb) p $r14
$16 = 0x404144 <main+7>
(gdb) p $r15
$17 = 0x404144 <main+7>
(gdb) p $r16
$18 = 0x404144 <main+7>
(gdb) p $r17
$19 = 0x404144 <main+7>
(gdb) p $r18
$20 = 0x404144 <main+7>
(gdb) p $r19
$21 = 0x404144 <main+7>
(gdb) p $r20
$22 = 0x404144 <main+7>
(gdb) p $r21
$23 = 0x404144 <main+7>
(gdb) p $r22
$24 = 0x404144 <main+7>
(gdb) p $r23
$25 = 0x404144 <main+7>
(gdb) p $r24
$26 = 0x404144 <main+7>
(gdb) p $r25
$27 = 0x404144 <main+7>
(gdb) p $r26
$28 = 0x404144 <main+7>
(gdb) p $r27
$29 = 0x404144 <main+7>
(gdb) p $r28
$30 = 0x404144 <main+7>
(gdb) p $r29
$31 = 0x404144 <main+7>
(gdb) quit
Program received signal SIGSEGV, Segmentation fault.
#0 0x404144 in ?? (main+7)
#1 0x404141 in ?? (main+2)
#2 0x404141 in ?? (main+2)
#3 0x404141 in ?? (main+2)
#4 0x404141 in ?? (main+2)
#5 0x404141 in ?? (main+2)
#6 0x404141 in ?? (main+2)
#7 0x404141 in ?? (main+2)
#8 0x404141 in ?? (main+2)
#9 0x404141 in ?? (main+2)
#10 0x404141 in ?? (main+2)
#11 0x404141 in ?? (main+2)
#12 0x404141 in ?? (main+2)
#13 0x404141 in ?? (main+2)
#14 0x404141 in ?? (main+2)
#15 0x404141 in ?? (main+2)
#16 0x404141 in ?? (main+2)
#17 0x404141 in ?? (main+2)
#18 0x404141 in ?? (main+2)
#19 0x404141 in ?? (main+2)
#20 0x404141 in ?? (main+2)
#21 0x404141 in ?? (main+2)
#22 0x404141 in ?? (main+2)
#23 0x404141 in ?? (main+2)
#24 0x404141 in ?? (main+2)
#25 0x404141 in ?? (main+2)
#26 0x404141 in ?? (main+2)
#27 0x404141 in ?? (main+2)
#28 0x404141 in ?? (main+2)
#29 0x404141 in ?? (main+2)
#30 0x404141 in ?? (main+2)
#31 0x404141 in ?? (main+2)
(gdb) break main
Breakpoint 1 at 0x404144: file main.c, line 19.
Starting program: ./nes/mii/mtftp -q
(gdb) p $pc
$1 = 0x404144 <main+7>
(gdb) p $r0
$2 = 0x404144 <main+7>
(gdb) p $r1
$3 = 0x404144 <main+7>
(gdb) p $r2
$4 = 0x404144 <main+7>
(gdb) p $r3
$5 = 0x404144 <main+7>
(gdb) p $r4
$6 = 0x404144 <main+7>
(gdb) p $r5
$7 = 0x404144 <main+7>
(gdb) p $r6
$8 = 0x404144 <main+7>
(gdb) p $r7
$9 = 0x404144 <main+7>
(gdb) p $r8
$10 = 0x404144 <main+7>
(gdb) p $r9
$11 = 0x404144 <main+7>
(gdb) p $r10
$12 = 0x404144 <main+7>
(gdb) p $r11
$13 = 0x404144 <main+7>
(gdb) p $r12
$14 = 0x404144 <main+7>
(gdb) p $r13
$15 = 0x404144 <main+7>
(gdb) p $r14
$16 = 0x404144 <main+7>
(gdb) p $r15
$17 = 0x404144 <main+7>
(gdb) p $r16
$18 = 0x404144 <main+7>
(gdb) p $r17
$19 = 0x404144 <main+7>
(gdb) p $r18
$20 = 0x404144 <main+7>
(gdb) p $r19
$21 = 0x404144 <main+7>
(gdb) p $r20
$22 = 0x404144 <main+7>
(gdb) p $r21
$23 = 0x404144 <main+7>
(gdb) p $r22
$24 = 0x404144 <main+7>
(gdb) p $r23
$25 = 0x404144 <main+7>
(gdb) p $r24
$26 = 0x404144 <main+7>
(gdb) p $r25
$27 = 0x404144 <main+7>
(gdb) p $r26
$28 = 0x404144 <main+7>
(gdb) p $r27
$29 = 0x404144 <main+7>
(gdb) p $r28
$30 = 0x404144 <main+7>
(gdb) p $r29
$31 = 0x404144 <main+7>
(gdb) quit
Program received signal SIGSEGV, Segmentation fault.
#0 0x404144 in ?? (main+7)
#1 0x404141 in ?? (main+2)
#2 0x404141 in ?? (main+2)
#3 0x404141 in ?? (main+2)
#4 0x404141 in ?? (main+2)
#5 0x404141 in ?? (main+2)
#6 0x404141 in ?? (main+2)
#7 0x404141 in ?? (main+2)
#8 0x404141 in ?? (main+2)
#9 0x404141 in ?? (main+2)
#10 0x404141 in ?? (main+2)
#11 0x404141 in ?? (main+2)
#12 0x404141 in ?? (main+2)
#13 0x404141 in ?? (main+2)
#14 0x404141 in ?? (main+2)
#15 0x404141 in ?? (main+2)
#16 0x404141 in ?? (main+2)
#17 0x404141 in ?? (main+2)
#18 0x404141 in ?? (main+2)
#19 0x404141 in ?? (main+2)
#20 0x404141 in ?? (main+2)
#21 0x404141 in ?? (main+2)
#22 0x404141 in ?? (main+2)
#23 0x404141 in ?? (main+2)
#24 0x404141 in ?? (main+2)
#25 0x404141 in ?? (main+2)
#26 0x404141 in ?? (main+2)
#27 0x404141 in ?? (main+2)
#28 0x404141 in ?? (main+2)
#29 0x404141 in ?? (main+2)
#30 0x404141 in ?? (main+2)
#31 0x404141 in ?? (main+2)
```

2ND EDITION

HACKING
THE ART OF EXPLOITATION

Super Mario World TAS Excerpt





Hackable Games

More than just speedrunning, other reasons to hack games:

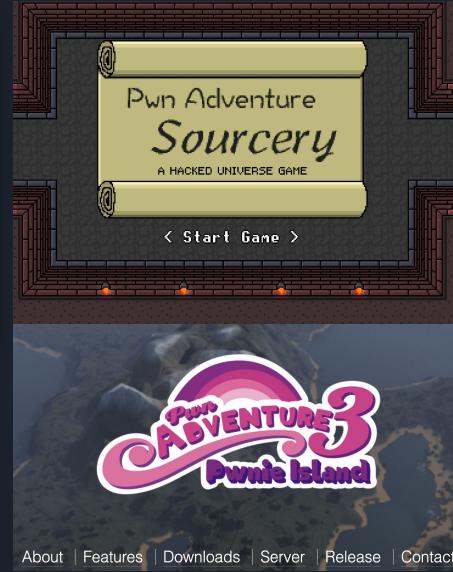
- For fun!
- For profit!
- For learning

Hackable Games

PwnAdventure

- [PwnAdventure2](#)
- [PwnAdventure3](#)
- [PwnAdventureZ \(NES\)](#)
- [SuperMonsterBall](#)
- [Sourcery \(Online\)](#)

Squally



[About](#) | [Features](#) | [Downloads](#) | [Server](#) | [Release](#) | [Contact](#)

Education and Training

- 4 years of college recommended
 - Computer Science
 - Computer Engineering
 - Electrical Engineering
- Preferred Languages: C/C++, Python
- College 4 year degree not required, but it opens extra doors. It's a great ROI
- Cyber Security Clubs
 - West Shore Computer Science Club
 - FITSec (Florida Tech)
 - HackUCF (UCF)
- Coding contests / challenges
 - Advent of Code
 - Game Jams
- Capture the Flag
 - Online cybersecurity competitions





Security as a Field

- Defense (bigger!)
 - Malware Analysis
 - Incident Response
 - Forensics
 - Network Security
- Offense (more fun!)
 - Network Penetration Testing
 - Vulnerability Research
 - CNO Dev



Government vs Commercial

Government

- ✗ Clearance usually required
- ✗ More offensive opportunities
- ✗ More stable
- ✗ Patriotism / Make a difference
- ✗ Less pay
- ✗ More geographically restricted

Commercial

- ✗ No clearance required
- ✗ Defensive focused
- ✗ Less Stable
- ✗ Less impactful
- ✗ Much higher pay opportunities
- ✗ More flexible location/WFH



Salary and Compensation

- New Grads with 4 year degree
 - \$90,000 - \$100,000 (\$42 - \$48 per hour)
 - Benefits
 - Health Insurance
 - 401K matching 5-10%
 - Yearly Bonus 10%
- Lots of different types of Cyber Security - Ethical Hacking / Offensive Cyber
 - Reverse Engineer
 - Vulnerability Researcher
 - CNO developer
- Companies: Raytheon CODEX, Research Innovations, Red Lattice, Cromulence, STR
- Cyber Security has typically had better benefits, raises, work perks, training than typical engineering firms



Other Factors

- Fun / Interesting Work
 - You get to hack things, and get paid for it
- This stuff matters
 - **NOT** annoying people with ads / stealing customer data to resell
 - **NOT** writing backend middleware connecting web sites to the cloud
 - You are helping your country / war-fighters
- You will never stop learning
 - Work is challenging / constantly solving new problems
 - Constantly learning new things
 - Tools / techniques you learn for cyber will make you a better developer
- Impact / Reward
 - Great feeling to land your exploit / crush a new piece of HW / SW



Attributions

- Youtube: [BotW Any% 23:42 \[WR\]](#) by Player5
- Youtube: [How to create the perfect speedrun](#) by Bismuth
- Youtube: [\[TAS\] Mario Kart 64 - All Cups -- 1P, GP, 150cc in 20:33:32](#) by weatherton
- Youtube: [Marl/O - Machine Learning for Video Games](#) by SethBling
- Book: [Hacking: The Art of Exploitation](#) by Jon Erickson
- Youtube: [Super Mario World TAS Arbitrary Code Execution Demo](#) by Masterjun3
- Micro500: <https://www.instructables.com/NESBot-Arduino-Powered-Robot-beating-Super-Mario-/>
- Youtube: [TASBot at DEF CON 24 - Robot Hacks Video Games full talk](#) by dwangoAC
- Youtube: [SNES Code Injection - Flappy Bird in SMW](#) by SethBling
-



Questions?!

