

FLARE-On 13 Challenge 9: NeonOutRun

By Mustafa Nasser (@mustafanasser)

Overview

NeonOutRun presents a retro-style arcade racing game wrapped in a modern desktop application. Players navigate a synthwave highway, clearing 28 consecutive laps to claim victory and uncover the hidden flag: **R3_ag3nt_4b0ut_t0_f331_d1zzy@flare-on.com**.

At its core, the application is built on the Tauri 2 framework, written in Rust, and embeds Google's V8 JavaScript engine via **rusty_v8** to execute the game's core physics and state logic. Rather than checking the flag in plain JavaScript or trivial native RPC handlers, the binary deeply interweaves the game runtime with compiled native hooks, multi-stage compression, and control-flow flattening.

Clearing all 28 laps normally only yields a decoy flag; to unlock the real flag, each lap must be finished with an exact, mathematically derived completion timing whose integer value represents the corresponding ASCII character of the true flag.



Figure 1: Decoy flag upon finishing normally (notice that it is *flare*, not *flare*)

To understand the challenge's architecture, the player must recover two distinct layers of resources: the frontend web assets (HTML5/Canvas UI and Tauri IPC bindings) and the embedded backend. While the frontend merely handles visual rendering and sends tick events via Tauri's custom asset protocol, the core physics, state machine, and lap timing computations are isolated inside an embedded V8 runtime.

Recovering the Frontend Assets

Tauri compresses all embedded web assets using Brotli, which can be located by searching the binary for `index.html`.

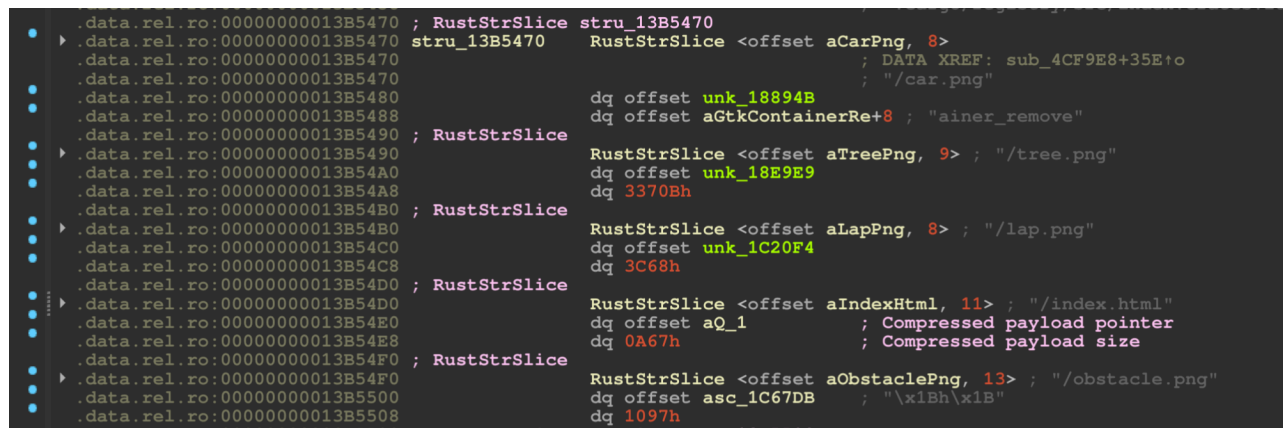


Figure 2: Frontend assets in the binary

Dumping, decompressing, and beautifying `index.html` yields the following:

```

<!DOCTYPE html>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0, maximum-scale=1.0,
user-scalable=no">
<style>

    * {
        user-select: none;
        -webkit-user-select: none;
        touch-action: none;
    }

    html,
    body {
        margin: 0;
        padding: 0;
        width: 100vw;
        height: 100vh;
        overflow: hidden;
        display: flex;
        background: #111;
        color: #0f0;
        font-family: monospace;
    }

```

```
#ui {
  width: 180px;
  height: 100%;
  background: #0d0d0d;
  padding: 15px;
  box-sizing: border-box;
  border-right: 2px solid #333;
  flex-shrink: 0;
  z-index: 10;
}

h1 {
  margin: 0 0 15px 0;
  font-size: 18px;
  color: #0f0;
}

p {
  font-size: 13px;
  line-height: 1.4;
  color: #aaa;
  margin: 6px 0;
}

.highlight {
  color: #0f0;
  font-weight: bold;
}

#canvas-container {
  flex-grow: 1;
  height: 100%;
  position: relative;
  background: #1b4d1b;
}

canvas {
  width: 100%;
  height: 100%;
  display: block;
}

.modal-screen {
  display: none;
  position: absolute;
  top: 40%;
  left: 50%;
  transform: translate(-50%, -50%);
  text-align: center;
  font-size: 20px;
  font-weight: bold;
  background: rgba(10, 10, 10, 0.94);
  padding: 25px 35px;
  border-radius: 10px;
  z-index: 20;
  max-width: 80%;
}

.red-modal {
  color: #ff3333;
  border: 3px solid #ff3333;
  box-shadow: 0 0 15px rgba(255, 0, 0, 0.5);
}
```

```

}

.yellow-modal {
  color: #f7e62e;
  border: 3px solid #f7e62e;
  box-shadow: 0 0 15px rgba(252, 223, 1, 0.5);
}

.green-modal {
  color: #00ff66;
  border: 3px solid #00ff66;
  box-shadow: 0 0 15px rgba(0, 255, 102, 0.5);
}
</style>
<div id="ui">
  <h1>Neon OutRun</h1>
  <p>Controls:</p>
  <p><span class="highlight">⬆️ / ⬇️</span> Accelerate</p>
  <p><span class="highlight">⬇️ / ⬆️</span> Steer</p>
  <p><span class="highlight">R</span> Restart</p> <br>
  <p>Lap: <span id="lap-num" class="highlight">1 / 28</span></p>
  <p>Cur. Lap: <span id="lap-timer" class="highlight">0.0s</span></p>
  <p>Prev. Lap: <span id="prev-lap" class="highlight">--</span></p>
  <p>Best Lap: <span id="best-lap" class="highlight">--</span></p>
</div>
<div id="canvas-container">
  <div id="crash-screen" class="modal-screen red-modal"> CRASHED<br><br> <span
    style="color:#fff; font-size:16px;">Press 'R' to Restart</span> </div>
  <div id="message-screen" class="modal-screen yellow-modal"> <span id="message"
    style="color:red; font-size:18px; font-weight:bold;
overflow-wrap:break-word;"></span><br><br> <span
    style="color:#fff; font-size:16px;">Press 'R' to Restart</span> </div>
  <div id="neon-screen" class="modal-screen green-modal"> <span id="neon"
    style="color:yellow; font-size:18px; font-weight:bold;
overflow-wrap:break-word;"></span><br><br> <span
    style="color:#fff; font-size:16px;">Press 'R' to Restart</span> </div> <canvas
id="neonCanvas"></canvas>
</div>    
<script>
  const { invoke } = window.__TAURI__.core;
  const canvas = document.getElementById('neonCanvas');
  const ctx = canvas.getContext('2d');
  const carImg = document.getElementById('carImg');
  const obsImg = document.getElementById('obsImg');
  const treeImg = document.getElementById('treeImg');
  const lapImg = document.getElementById('lapImg');
  const crashScreen = document.getElementById('crash-screen');
  const messageScreen = document.getElementById('message-screen');
  const neonScreen = document.getElementById('neon-screen');
  let clientLaps = [];
  const trees = [];
  function generateTrees() {
    trees.length = 0;
    for (let y = 5000; y >= -105000; y -= 20 + Math.random() * 15) {
      const numLeft = 1 + Math.floor(Math.random() * 2);
      for (let k = 0; k < numLeft; k++) {
        trees.push({
          side: 'left',
          relX: Math.random(),

```

```

        worldY: y + (Math.random() * 20 - 10),
        scale: 0.75 + Math.random() * 0.5
    });
}
const numRight = 1 + Math.floor(Math.random() * 2);
for (let k = 0; k < numRight; k++) {
    trees.push({
        side: 'right',
        relX: Math.random(),
        worldY: y + (Math.random() * 20 - 10),
        scale: 0.75 + Math.random() * 0.5
    });
}
}
trees.sort((a, b) => a.worldY - b.worldY);
}
generateTrees();
function resizeCanvas() {
    canvas.width = canvas.clientWidth;
    canvas.height = canvas.clientHeight;
}
resizeCanvas();
window.addEventListener('resize', resizeCanvas);
const AudioContext = window.AudioContext || window.webkitAudioContext;
const actx = new AudioContext();
const osc = actx.createOscillator();
const gain = actx.createGain();
osc.type = 'sawtooth';
osc.connect(gain);
gain.connect(actx.destination);
gain.gain.value = 0;
osc.start();
const input = {
    up: false,
    down: false,
    left: false,
    right: false,
    restart: false,
    bounds: null
};
let audioStarted = false;
window.addEventListener('keydown', e => {
    if (!audioStarted) {
        actx.resume();
        audioStarted = true;
    }
    if (e.key === 'ArrowUp')
        input.up = true;
    if (e.key === 'ArrowDown')
        input.down = true;
    if (e.key === 'ArrowLeft')
        input.left = true;
    if (e.key === 'ArrowRight')
        input.right = true;
    if (e.key === 'r' || e.key === 'R')
        input.restart = true;
});
window.addEventListener('keyup', e => {
    if (e.key === 'ArrowUp')
        input.up = false;
    if (e.key === 'ArrowDown')
        input.down = false;
});

```

```

    if (e.key === 'ArrowLeft')
      input.left = false;
    if (e.key === 'ArrowRight')
      input.right = false;
    if (e.key === 'r' || e.key === 'R')
      input.restart = false;
  });
  let isInit = false;
  async function initNeon() {
    if (isInit)
      return;
    isInit = true;
    loop();
  }
  let inFrame = false;
  async function loop() {
    if (inFrame) {
      requestAnimationFrame(loop);
      return;
    }
    inFrame = true;
    try {
      const roadWidth = Math.min(320, canvas.width * 0.7);
      const roadLeft = (canvas.width - roadWidth) / 2;
      const roadRight = roadLeft + roadWidth;
      const roadCenter = canvas.width / 2;
      input.bounds = {
        left: roadLeft,
        right: roadRight
      };
      let resStr = await invoke('plugin:neon|e', { input: JSON.stringify(input) });
      let res = JSON.parse(resStr);
      let state = res.state;
      let obstacles = res.obstacles || [];
      if (res.laps)
        clientLaps = res.laps;
      ctx.clearRect(0, 0, canvas.width, canvas.height);
      ctx.fillStyle = '#1b4d1b';
      ctx.fillRect(0, 0, canvas.width, canvas.height);
      ctx.fillStyle = '#444';
      ctx.fillRect(roadLeft, 0, roadWidth, canvas.height);
      ctx.setLineDash([]);
      ctx.strokeStyle = '#e00';
      ctx.lineWidth = 5;
      ctx.beginPath();
      ctx.moveTo(roadLeft, 0);
      ctx.lineTo(roadLeft, canvas.height);
      ctx.moveTo(roadRight, 0);
      ctx.lineTo(roadRight, canvas.height);
      ctx.stroke();
      const carScreenY = canvas.height * state.carRatio;
      ctx.strokeStyle = '#fff';
      ctx.lineWidth = 8;
      ctx.setLineDash([40, 40]);
      ctx.lineDashOffset = state.y;
      ctx.beginPath();
      ctx.moveTo(roadCenter, 0);
      ctx.lineTo(roadCenter, canvas.height);
      ctx.stroke();
      for (let i = 0; i < clientLaps.length; i++) {
        const lapY = clientLaps[i];
        const screenY = lapY - state.y + carScreenY;

```

```

    if (screenY >= -60 && screenY <= canvas.height + 60) {
      const w = 48;
      const h = 48;
      if (lapImg && lapImg.complete && lapImg.naturalWidth > 0) {
        ctx.drawImage(lapImg, roadCenter - w / 2, screenY - h / 2, w, h);
      } else {
        ctx.fillStyle = '#ff0';
        ctx.beginPath();
        ctx.arc(roadCenter, screenY, 20, 0, Math.PI * 2);
        ctx.fill();
      }
    }
  }
}
for (let i = 0; i < trees.length; i++) {
  const t = trees[i];
  const screenY = t.worldY - state.y + carScreenY;
  if (screenY >= -100 && screenY <= canvas.height + 100) {
    let treeX = 0;
    if (t.side === 'left') {
      treeX = t.relX * Math.max(10, roadLeft - 30);
    } else {
      treeX = roadRight + 25 + t.relX * Math.max(10, canvas.width - roadRight -
35);
    }
    const w = 50 * t.scale;
    const h = 60 * t.scale;
    if (treeImg && treeImg.complete && treeImg.naturalWidth > 0) {
      ctx.drawImage(treeImg, treeX - w / 2, screenY - h / 2, w, h);
    } else {
      ctx.fillStyle = '#0e4a0e';
      ctx.beginPath();
      ctx.arc(treeX, screenY, 20 * t.scale, 0, Math.PI * 2);
      ctx.fill();
    }
  }
}
for (let i = 0; i < obstacles.length; i++) {
  const obs = obstacles[i];
  const screenY = obs.worldY - state.y + carScreenY;
  if (screenY >= -50 && screenY <= canvas.height + 50) {
    const obsX = roadLeft + obs.relX * (roadRight - roadLeft);
    const w = 42;
    const h = 42;
    if (obsImg && obsImg.complete && obsImg.naturalWidth > 0) {
      ctx.drawImage(obsImg, obsX - w / 2, screenY - h / 2, w, h);
    } else {
      ctx.fillStyle = '#f00';
      ctx.beginPath();
      ctx.arc(obsX, screenY, 18, 0, Math.PI * 2);
      ctx.fill();
    }
  }
}
}
ctx.save();
ctx.translate(state.x, carScreenY);
ctx.rotate(state.angle);
if (carImg.complete && carImg.naturalWidth > 0) {
  ctx.drawImage(carImg, -25, -25, 50, 50);
} else {
  ctx.fillStyle = '#00f';
  ctx.fillRect(-25, -25, 50, 50);
}
}

```

```

    ctx.restore();
    document.getElementById('lap-num').innerText = `${state.currentLap} / ${res.totalLaps}
|| 28}`;
    document.getElementById('lap-timer').innerText = state.currentLapTime || '0.0s';
    document.getElementById('prev-lap').innerText = state.previousLapTime || '--';
    document.getElementById('best-lap').innerText = state.bestLapTime || '--';
    if (state.crashed) {
        crashScreen.style.display = 'block';
        messageScreen.style.display = 'none';
        neonScreen.style.display = 'none';
    } else if (state.blocked) {
        crashScreen.style.display = 'none';
        messageScreen.style.display = 'block';
        neonScreen.style.display = 'none';
        document.getElementById('message').innerText = state.message;
    } else if (state.finished) {
        crashScreen.style.display = 'none';
        messageScreen.style.display = 'none';
        neonScreen.style.display = 'block';
        document.getElementById('neon').innerText = state.neon;
    } else {
        crashScreen.style.display = 'none';
        messageScreen.style.display = 'none';
        neonScreen.style.display = 'none';
    }
    if (audioStarted && !state.crashed && !state.blocked && !state.finished) {
        if (Math.abs(state.speed) > 0.05) {
            gain.gain.value = 0.05;
            osc.frequency.value = 50 + Math.abs(state.speed) * 30;
        } else {
            gain.gain.value = 0;
        }
    } else if (state.crashed) {
        gain.gain.value = 0.08;
        osc.frequency.value = 35;
    } else {
        gain.gain.value = 0;
    }
} catch (e) {
    console.error(e);
} finally {
    inFrame = false;
}
requestAnimationFrame(loop);
}
setTimeout(() => {
    if (!isInit)
        initNeon();
}, 500);
if (carImg.complete) {
    initNeon();
} else {
    carImg.onload = initNeon;
}
</script>

```

This is the client-side presentation layer of the game, holding the HTML5/CSS elements (lap counters and modal status screens), a 2D canvas renderer that draws the road and roadside sprites based on backend-provided coordinates, a basic Web Audio synthesizer, and the input listener.

We can safely ignore the visual rendering routines and focus entirely on the single IPC bridge inside the game loop: `await invoke('plugin:neon|e', { input: JSON.stringify(input) })`. This call proves that the frontend delegates 100% of state progression, physics, and validation to the native Tauri plugin `neon` via command `e`.

Recovering the Backend JS

The backend is written in JavaScript and loaded using an embedded V8 engine; it is XOR-encrypted and compressed using LZSS, ZSTD, and DEFLATE.

To recover it, find the function `v8::ScriptCompiler::Compile` by searching for its string reference, set a breakpoint, and run the debugger until hitting it. A pointer to the backend JS is in the `RBX` register or in the dereference of the second argument (the `Source` object).

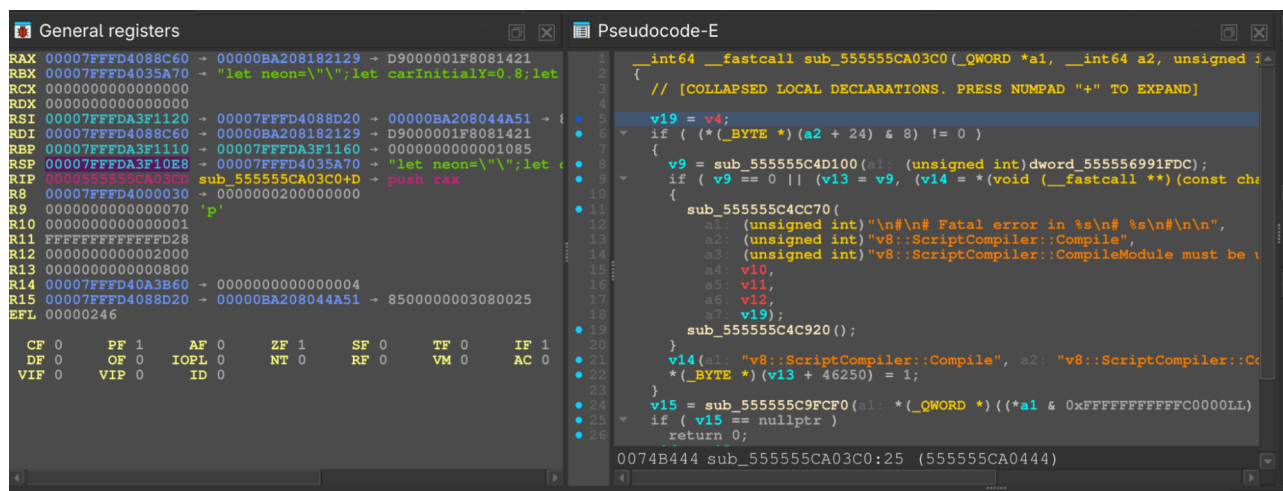


Figure 3: Backend JS in memory at `v8::ScriptCompiler::Compile`

Dumping and beautifying it yields the following:

```
let neon = '';
let carInitialY = 0.8;
let obstacles = [];
let laps = [];
let state = {
  x: 0,
  y: 0,
  speed: 0,
  angle: 0,
  score: 0,
  neon: '',
  crashed: false,
```

```

    blocked: false,
    finished: false,
    initialized: false,
    currentLap: 1,
    totalLaps: 28,
    currentLapTime: '0.0s',
    previousLapTime: '--',
    bestLapTime: '--',
    bestLapSeconds: Infinity,
    lapStartTime: 0,
    carRatio: carInitialY
  };
  const messages = [
    'I bled, so the world must bleed with me',
    'My agony is everyone's inheritance',
    'I suffered the fire, now I'm passing the torch',
    'Shared pain is the only true justice',
    'I hurt. Now it's your turn',
    'My scars, your sentence',
    'Misery loves company, and I'm hosting',
    'A taste of my torment for everyone',
    'The flames that burned me will reduce this world to ash',
    'I was handed a cup of poison, and I intend to pour it for the world',
    'Once the victim of the storm, now I am the rain',
    'I took the damage; the world is paying the toll',
    'I didn't heal, I just weaponized it',
    'Pain was my teacher, and class is in session'
  ];
  map = [5, 4, 32, 5, 9, 8, 6, 24, 2, 12, 3, 9, 2, 5, 4, 5, 2, 7, 1, 38, 16, 4, 23, 29, 5, 1, 2, 12];
  function generateLaps() {
    laps = [];
    let splits = [];
    for (let i = 0; i < 27; i++) {
      splits.push(Math.round(-1000 - Math.random() * 98000));
    }
    splits.push(-100000);
    splits.sort((a, b) => b - a);
    for (let i = 0; i < splits.length; i++) {
      let prevY = i === 0 ? 0 : splits[i - 1];
      if (prevY - splits[i] < 800) {
        splits[i] = prevY - 800;
      }
    }
    laps = splits;
  }
  function generateObstacles() {
    obstacles = [];
    let y = -400;
    let lastLapY = laps.length > 0 ? laps[laps.length - 1] : -100000;
    while (y >= lastLapY) {
      let nearLap = false;
      for (let f = 0; f < laps.length; f++) {
        if (Math.abs(y - laps[f]) < 100) {
          nearLap = true;
          break;
        }
      }
      if (!nearLap) {
        let relX = 0.15 + Math.random() * 0.7;
        obstacles.push({
          relX: relX,
          worldY: y
        });
      }
      y += 100;
    }
  }

```

```

    });
  }
  let absY = Math.abs(y) / 100000;
  let progress = absY > 1 ? 1 : absY;
  let baseStep = 220 - progress * 170;
  let randStep = 200 - progress * 140;
  let step = baseStep + Math.random() * randStep;
  y -= step;
}
function tick(input_json) {
  let input = JSON.parse(input_json);
  roadLeft = input.bounds.left;
  roadRight = input.bounds.right;
  if (!state.initialized) {
    state.x = (roadLeft + roadRight) / 2;
    generateLaps();
    generateObstacles();
    state.initialized = true;
  }
  if (!state.crashed && !state.blocked && !state.finished) {
    let now = Date.now();
    if (!state.lapStartTime && (input.up || input.down || input.left || input.right)) {
      state.lapStartTime = now;
    }
    if (state.lapStartTime) {
      state.currentLapTime = ((now - state.lapStartTime) / 1000).toFixed(1) + 's';
    }
    if (input.up)
      state.speed += 0.5;
    if (input.down)
      state.speed -= 0.5;
    state.speed *= 0.96;
    if (state.speed > 0.1 || state.speed < -0.1) {
      if (input.left)
        state.angle -= 0.05;
      if (input.right)
        state.angle += 0.05;
    }
    state.x += Math.sin(state.angle) * state.speed;
    state.y -= Math.cos(state.angle) * state.speed;
    let ratio = carInitialY;
    if (state.y < 0) {
      ratio = Math.max(0.05, carInitialY * (1 + state.y / 90000));
    }
    state.carRatio = ratio;
    if (state.y >= 4000 && !state.blocked) {
      state.blocked = true;
      state.message = messages[Math.floor(Math.random() * messages.length)];
      state.speed = 0;
    }
    if (state.currentLap <= 28) {
      let targetY = laps[state.currentLap - 1];
      if (state.y <= targetY) {
        state.previousLapTime = state.currentLapTime;
        let lapSecs = (now - state.lapStartTime) / 1000;
        state.bestLapSeconds = Math.min(state.bestLapSeconds, lapSecs);
        state.bestLapTime = state.bestLapSeconds.toFixed(1) + 's';
        neon += messages[Math.floor((state.currentLap - 1) /
2)][map[state.currentLap - 1]];
        if (state.currentLap < 28) {
          state.currentLap++;

```

```

        state.lapStartTime = now;
    } else {
        neon += String.fromCharCode(64, 102, 49, 97, 114, 101, 45, 111,
110, 46, 99, 111, 109);
        state.finished = true;
        state.neon = neon.replaceAll(' ', '_');
        state.speed = 0;
    }
}
}
if (state.x < roadLeft + 25 || state.x > roadRight - 25) {
    state.crashed = true;
    state.speed = 0;
}
let roadW = roadRight - roadLeft;
for (let i = 0; i < obstacles.length; i++) {
    let obs = obstacles[i];
    let obsX = roadLeft + obs.relX * roadW;
    let dx = state.x - obsX;
    let dy = state.y - obs.worldY;
    let distSq = dx * dx + dy * dy;
    if (distSq < 1024) {
        state.crashed = true;
        state.speed = 0;
        break;
    }
}
}
if (input.restart) {
    let currentBestTime = state.bestLapTime || '--';
    let currentBestSecs = state.bestLapSeconds || Infinity;
    neon = '';
    state = {
        x: (roadLeft + roadRight) / 2,
        y: 0,
        speed: 0,
        angle: 0,
        score: 0,
        message: '',
        neon: '',
        crashed: false,
        blocked: false,
        finished: false,
        initialized: true,
        currentLap: 1,
        totalLaps: 28,
        currentLapTime: '0.0s',
        previousLapTime: '--',
        bestLapTime: currentBestTime,
        bestLapSeconds: currentBestSecs,
        lapStartTime: 0,
        carRatio: carInitialY
    };
    generateLaps();
    generateObstacles();
}
return JSON.stringify({
    state: state,
    obstacles: obstacles,
    laps: laps,
    totalLaps: 28
});

```

}

This JS serves as the backend game engine executed inside the isolated V8 runtime, managing vehicle kinematics, road boundaries, collision detection, procedural obstacle/lap generation, and the `tick(input_json)` state update loop. It tracks player movement and elapsed lap times, and upon lap completion, incrementally builds a decoy victory string (`neon`) by indexing characters from an array of hardcoded thematic quote messages before appending `@flare-on.com` on the final lap. You may modify and replace the backend JS in-memory to increase the car speed and disable obstacle crashing.

The Tauri plugin `neon` command `e` forwards the inputted JSON state to `tick(input_json)` in the backend JS. This can be verified by setting a breakpoint at `v8::Function::Call` and running the debugger until hitting it; the input JSON can be found by dereferencing the `R8` register.

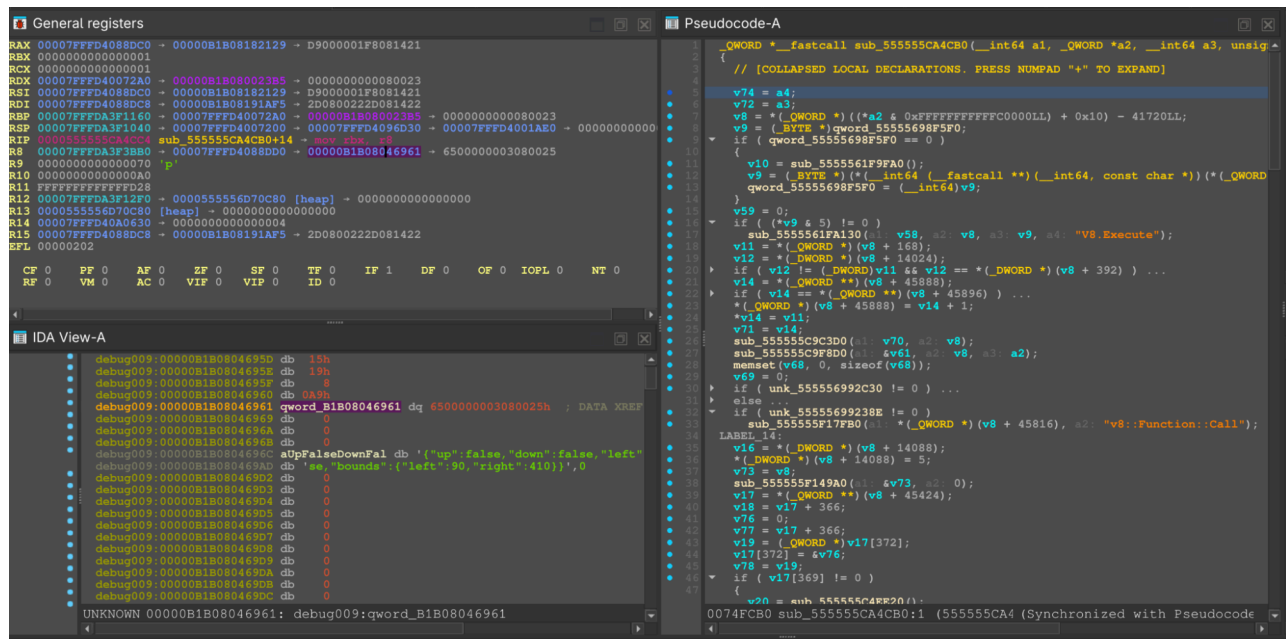


Figure 4: Calling the backend JS function tick

Unraveling the V8 Hooks

At this stage of the investigation, looking at both the frontend and backend JS reveals zero visible cryptographic logic or references to a real flag. This absence leaves only one logical conclusion: the game's true validation mechanism is hidden at the boundary between the JavaScript runtime and the native binary.

Because the game runs inside an embedded V8 engine hosted by a compiled Rust backend, the native Rust code silently intercepts and replaces standard JavaScript built-in functions in memory. Instead of performing plain calculations, these hooked functions quietly feed the player's lap actions into the native Rust backend, allowing the binary to secretly perform mathematical verification and dynamically overwrite the decoy flag with the true flag only when the exact winning conditions are satisfied. The JavaScript from the frontend is not

affected because it runs inside the OS webview (WebKitGTK on Linux), which uses its own JavaScriptCore engine.

Specifically, three built-in JS functions are hooked:

- `Array.prototype.sort` → Function at RVA `0x67DEB2`
- `Math.min` → Function at RVA `0x67EB83`
- `String.prototype.replaceAll` → Function at RVA `0x67E81E`

You can verify these hooks by inspecting the hook registration function at RVA `0x688C64`:

```

64     v505.m256i_i64[0] = 6;
65     v505.m256i_i64[1] = v18;
66     v505.m256i_i64[2] = 6;
67     for ( i = 0; i != 6; ++i )                // Decrypting "String"
68     {
69         v20 = *(sub_555555BE0E34)(a1: v494, a2: 6, a3: i, a4: &stru_55555691B358);
70         v21 = (sub_555555BE0E34)(a1: v18, a2: 6, a3: i, a4: &stru_55555691B358);
71         *v21 ^= v20;
72     }
73     sub_555555A16C31(a1: v528, a2: &v505);
74     if ( !_OFSUB__(0, v528[0].m256i_i64[0]) ) ...
75     v25 = *&v528[0].m256i_u64[1];
76     v26 = v528[0].m256i_i64[3];
77     sub_555555B55B54(a1: 6, a2: v494, a3: v22, a4: v23, a5: v24);
78     v27 = sub_555555B6C34D(a1: v501, a2: *(&v25 + 1), a3: v26);
79     sub_555555B55B54(a1: v25, a2: *(&v25 + 1), a3: v28, a4: v29, a5: v30);
80     if ( v27 != 0 )
81     {
82         v31 = (sub_555555B6C285)(a1: *(&v495 + 1), a2: v501, a3: v27);
83         if ( v31 != 0 )
84         {
85             v32 = v31;
86             if ( (sub_555555C4A170)(a1: v31) != 0 )
87             {
88                 *&v495 = v32;
89                 v33 = sub_555555BDCC37(alignment: 1u, size: 9u);
90                 *v33 = 0xD2102CFC48F6CAEALL;
91                 v494 = v33;
92                 *(v33 + 8) = -104;
93                 v34 = sub_555555BDCC37(alignment: 1u, size: 9u);
94                 *v34 = 0xA26958933C99B89ALL;
95                 *(v34 + 8) = -3;
96                 v499 = 9;
97                 v500.m256i_i64[0] = v34;
98                 v500.m256i_i64[1] = 9;
99                 for ( j = 0; j != 9; ++j )                // Decrypting "prototype"
100                 {
101                     v36 = *(sub_555555BE0E34)(a1: v494, a2: 9, a3: j, a4: &stru_55555691B370);
102                     v37 = (sub_555555BE0E34)(a1: v34, a2: 9, a3: j, a4: &stru_55555691B370);
103                     *v37 ^= v36;
104                 }
105                 sub_555555A16C31(a1: v528, a2: &v499);

```

Figure 5: Registering native V8 hooks at RVA `0x688C64` (Part 1)

```

96     v499 = 9;
97     v500.m256i_i64[0] = v34;
98     v500.m256i_i64[1] = 9;
99     for ( j = 0; j != 9; ++j )                // Decrypting "prototype"
100     {
101         v36 = *(sub_555555BE0E34)(a1: v494, a2: 9, a3: j, a4: &stru_55555691B370);
102         v37 = (sub_555555BE0E34)(a1: v34, a2: 9, a3: j, a4: &stru_55555691B370);
103         *v37 ^= v36;
104     }
105     sub_555555A16C31(a1: v528, a2: &v499);
106     if ( v528[0].m256i_i64[0] != 0x8000000000000000LL ) ...
107     v41 = *&v528[0].m256i_u64[1];
108     v42 = v528[0].m256i_i64[3];
109     sub_555555B55B54(a1: 9, a2: v494, a3: v38, a4: v39, a5: v40);
110     v43 = v501;
111     v44 = sub_555555B6C34D(a1: v501, a2: *(&v41 + 1), a3: v42);
112     sub_555555B55B54(a1: v41, a2: *(&v41 + 1), a3: v45, a4: v46, a5: v47);
113     if ( v44 != 0 )
114     {
115         v48 = (sub_555555B6C285)(a1: v495, a2: v43, a3: v44);
116         if ( v48 != 0 )
117         {
118             v49 = v48;
119             if ( (sub_555555C4A170)(a1: v48) != 0 )
120             {
121                 *&v495 = v49;
122                 v50 = sub_555555BDCC37(alignment: 1u, size: 0xAu);
123                 *v50 = 0x7F2C0300298E46F3LL;
124                 v494 = v50;
125                 *(v50 + 4) = 16054;
126                 v51 = sub_555555BDCC37(alignment: 1u, size: 0xAu);
127                 *v51 = 0x3E49606145FE2381LL;
128                 *(v51 + 8) = 21210;
129                 v497.m256i_i64[0] = 10;
130                 v497.m256i_i64[1] = v51;
131                 v497.m256i_i64[2] = 10;
132                 for ( k = 0; k != 10; ++k )    // Decrypring "replaceAll"
133                 {
134                     v53 = *(sub_555555BE0E34)(a1: v494, a2: 10, a3: k, a4: &stru_55555691B370);
135                     v54 = (sub_555555BE0E34)(a1: v51, a2: 10, a3: k, a4: &stru_55555691B370);
136                     *v54 ^= v53;
137                 }

```

Figure 6: Registering native V8 hooks at RVA 0x688C64 (Part 2)

```

128      *(v51 + 8) = 21210;
129      v497.m256i_i64[0] = 10;
130      v497.m256i_i64[1] = v51;
131      v497.m256i_i64[2] = 10;
132      for ( k = 0; k != 10; ++k )          // Decrypring "replaceAll"
133      {
134          v53 = *(sub_555555BE0E34)(a1: v494, a2: 10, a3: k, a4: &stru_55555691B388);
135          v54 = (sub_555555BE0E34)(a1: v51, a2: 10, a3: k, a4: &stru_55555691B388);
136          *v54 ^= v53;
137      }
138      sub_555555A16C31(a1: v528, a2: &v497);
139      if ( v528[0].m256i_i64[0] != 0x8000000000000000LL )
140      {
141          qmemcpy(v529, v528, 40);
142          goto LABEL_574;
143      }
144      v58 = *&v528[0].m256i_u64[1];
145      v59 = v528[0].m256i_i64[3];
146      sub_555555B55B54(a1: 10, a2: v494, a3: v55, a4: v56, a5: v57);
147      v60 = v501;
148      v61 = sub_555555B6C34D(a1: v501, a2: *(&v58 + 1), a3: v59);
149      sub_555555B55B54(a1: v58, a2: *(&v58 + 1), a3: v62, a4: v63, a5: v64);
150      if ( v61 != 0 )
151      {
152          v65 = (sub_555555B6C285)(a1: v495, a2: v60, a3: v61);
153          if ( v65 != 0 )
154          {
155              v66 = v65;
156              if ( (sub_555555C4A150)(a1: v65) != 0 )
157              {
158                  v67 = sub_555555B6BA48(a1: v60);
159                  v68 = (sub_555555BE0E4C)(a1: v67, a2: v66);
160                  v70 = v69;
161                  v529[0].m256i_i64[0] = v68;
162                  v529[0].m256i_i64[1] = v69;
163                  v71 = (sub_555555BD2B09)(a1: 0);
164                  if ( v71 == 0 )
165                      goto LABEL_610;
166                  v72 = (sub_555555A735CC)(a1: v71, a2: &stru_55555691B3A0);
167                  v74 = v73;
168                  (sub_555555BDBEFE)(a1: v72);
169                  *v72 = v68;
170                  v72[1] = v70;
171                  ++*v74;
172                  v529[0].m256i_i64[2] = sub_555555BD281E; // Hook function for
173                                                          // String.prototype.replaceAll
174                  v529[0].m256i_i32[6] = 0;
175                  *v529[0].m256i_i8 = 0;
176                  *(&v529[0].m256i_i64[3] + 4) = 0x1000000000LL;
177                  v75 = (sub_555555B6C627)(a1: v529, a2: v60);
178                  if ( v75 != 0 )
179                      (sub_555555B6C2B7)(a1: v495, a2: v60, a3: v61, a4: v75);
180              }
181          }
182      }

```

Figure 7: Hooking String.prototype.replaceAll by the function at RVA 0x688C64

You can also verify this dynamically by setting a breakpoint on `v8::Function::New`, which registers the hook functions within the V8 context (the hook function pointer is passed in the `RSI` register). Each hook function accepts a single pointer to a structure containing the V8 scope handle, `v8::FunctionCallbackArguments`, and a return value slot.

From the backend JS, each hooked function is called as follows:

- `Array.prototype.sort` → Called once per race initialization with a random list of 28 negative integers (for the 28 laps).
- `Math.min` → Called once upon completing each lap with two arguments: `state.bestLapSeconds` and `lapSecs` (the current lap time).

- `String.prototype.replaceAll` → Called once after finishing lap 28 with the decoy flag to replace all spaces with `_`.

The Hook for `Array.prototype.sort`

- Calls the original `Array.prototype.sort` and stores the result in the return object.
- Generates a random positive scale and multiplies it by the 28 input integers.
- Saves the scaled array in a global vector.
- Calculates the summation of the scaled array and saves it in a global variable.

```

1  //
2  // Hook for Array.prototype.sort
3  //
4  __int64 __fastcall sub_67DEB2(_QWORD *a1)
5  {
6      // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
7
8      v2 = sub_67E4F2((__int64)a1);
9      v71 = v2;
10     v72 = a1;
11     v70 = sub_6F6550(a1);
12     v3 = sub_618594(a1);
13     v4 = sub_67E50C(a1: &v71, a2: sub_67E771, a3: v3, a4: &v72); // Calling the original Array.prototype.sort
14     if ( v4 == 0 ) ...
15     sub_6F65A0(a1: &v70, a2: v4);
16     if ( (unsigned __int8)sub_6F6160() == 0 ) ...
17     v55 = v3;
18     v5 = sub_6F62E0(a1: v3); // Generating a random scale
19
20     std::sys::pal::unix::time::Timespec::now::hdc5a35cbb551aac();
21     v73[0] = v6;
22     v74[0] = 0;
23     LODWORD(v73[1]) = v7;
24     LODWORD(v74[1]) = 0;
25     sub_6464D0(a1: (__int64)&v61, a2: v73, a3: v74);
26     v8 = 42;
27     if ( (_BYTE)v61 == 0 ) ...
28     if ( __readfsbyte(0xFFFFF70) == 0 ) ...
29     v66 = v2;
30     __writefsqword(0xFFFFF68, 0x5851F42D4C957F2DLL * (__readfsqword(0xFFFFF68) + v8) + 0x14057B7EF767814FLL);
31     compiler_builtins::math::full_availability::round::round();
32     v58 = v9;
33     v61 = sub_67DE74(a1: v5, a2: 8); // The random scale value
34     v62 = v10;
35     v63 = 0;
36     v11 = 0;
37     if ( v61 < v5 ) ...
38     else ...
39     v14 = v66;
40     while ( v5 != v11 )
41     {
42         sub_617A57(a1: v14);
43         v15 = sub_617F4C(a1: v14);
44         v16 = sub_6F62A0(a1: v55, a2: v15, a3: v11);
45         v17 = 0.0;
46         if ( v16 != 0 )
47         {
48             v18 = sub_61821D(a1: v16, a2: v14);
49             v17 = 0.0;
50             if ( v18 != 0 )
51             {
52                 v19.m128d_f64[0] = sub_6F62F0(a1: v18); // Multiplying the scale by all the integers
53                 v17 = COERCE_DOUBLE(*(_QWORD *)&_mm_cmpord_sd(v19, v19).m128d_f64[0] & *(_QWORD *)&v19.m128d_f64[0]) * v58;
54             }
55         }
56         v12[v13++] = v17; // Saving the scaled integers
57         ++v11;
58     }
59 }
60

```

Figure 8: The hook for `Array.prototype.sort` (Part 1)

```

56     }
57     }
58     v12[v13++] = v17; // Saving the scaled integers
59     ++v11;
60 }
61 v67 = v61;
62 v20 = 0.0;
63 v21 = 0;
64 for ( i = 801022094; ; i ^= v21 )
65 {
66     if ( i == -953704350 )
67     {
68         v56 = v20;
69         sub_5DFF31(a1: (__int64)&v61);
70         v25 = (volatile __int32 *)v62;
71         if ( (_BYTE)v61 == 0 )
72         {
73             v62[1] = 0.0;
74             sub_69C4A0(a1: v25);
75             v21 = -1926574070;
76             goto LABEL_61;
77         }
78         if ( i != 801022094 )
79             break;
80         v21 = -392625940;
81         if ( v13 != 0 )
82         {
83             v31 = nullptr;
84             v20 = -0.0;
85             do // Calculating sum for the scaled integers
86             {
87                 v20 = v20 + v12[(__QWORD)v31];
88                 v31 = (char *)v31 + 1;
89             } while ( (void *)v13 != v31 );
90         }
91         else
92         {
93             v20 = -0.0;
94         }
95     LABEL_62:
96     ;
97     }
98     switch ( i )
99     {
100     case 1242394728:
101         v56 = v20;
102         sub_604A82(a1: (__int64)&v61, a2: dword_1438590);
103         v23 = (volatile __int32 *)v62;
104         if ( (_BYTE)v61 == 0 )
105         {
106             v62[1] = 0.0;
107             sub_69C4A0(a1: v23);
108             v21 = 1054384928;
109             goto LABEL_61;
110         }
111         case 1573973377:
112             v56 = v20;
113             sub_67E679(a1: (__int64)&v61); // Locking access to the global
114                                           // vector holding the array of
115                                           // the scaled integers
116     }
117     v26 = v62;

```

Figure 9: The hook for Array.prototype.sort (Part 2)

```

108     goto LABEL_61;
109     case 1573973377:
110         v56 = v20;
111         sub_67E679(a1: (__int64)&v61); // Locking access to the global
112                                           // vector holding the array of
113                                           // the scaled integers
114     }
115     v26 = v62;
116     if ( (_BYTE)v61 == 0 )
117     {
118         if ( *((__QWORD *)v62 + 4) != 0 )
119         {
120             sub_5D5B57();
121             v26[4] = 0.0;
122         }
123         v26[3] = 0.0;
124         v28 = sub_67DE74(a1: v13, a2: 8);
125         v29 = v27;
126         if ( v13 != 0 )
127         { // Saving the array of integers in the global vector
128             memcpy(dest: v27, src: v12, n: 8 * v13);
129             v30 = v13;
130         }
131         else
132         {
133             v30 = 0;
134         }

```

Figure 10: The hook for Array.prototype.sort (Part 3)

```

233 if ( i != -239533400 )
234     goto LABEL_62;
235 v57 = v20;
236 sub_604A82(a1: (__int64)&v61, a2: dword_1438580); // Locking access to the global
237 // variable holding the sum of
238 // the scaled integers
239
240 v53 = (volatile __int32 *)v62;
241 if ( (_BYTE)v61 == 0 )
242     v62[1] = v57; // Saving the sum value
243 sub_69C4A0(a1: v53);
244 sub_6120FA(a1: v67, a2: (__int64)v12);
245 v2 = v66;
246 return sub_68CFCE(a1: v2);
247 }

```

Figure 11: The hook for Array.prototype.sort (Part 4)

The Hook for Math.min

- Calls the original `Math.min` and stores the result in the return object.
- Reads the second argument (the current lap time) along with the current lap index (from a global incremental counter).
- Pops the front value from the global vector of scaled integers.
- Hashes the lap index using an 8-byte hashing algorithm.
- Calculates a shift from the hash using a separate function at RVA `0x68003E`.
- Converts the current lap time integer into an ASCII character and appends it to a global string representing the candidate real flag.
- Computes `integer * sin(shift ± lap_time_u8)` and adds the result to a global accumulator:
 - The sine function is implemented using a 16th-degree Taylor series expansion.
 - The input `shift ± lap_time_u8` is converted from degrees to radians.
 - The sign before `lap_time_u8` is negative (-) if `shift` is even, and positive (+) if `shift` is odd.

```

1  //
2  //
3  // Hook for Math.min
4  //
5  //
6  __int64 __fastcall sub_67EB83(__int64 a1)
7  {
8      // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
9
10     v2 = sub_67E4F2(a1);
11     v80 = v2;
12     v81 = a1;
13     v82 = sub_6F6550((__QWORD *)a1);
14     v3 = (unsigned __int64 *)sub_687E7D(a1: 0);
15     if ( v3 == nullptr )
16     {
17         std[cb0c3ada4437c7f6]::thread::local::panic_access_error();
18     }
19     v4 = v3;
20     v5 = *v3;
21     if ( v5 >= 0x7FFFFFFFFFFFFFFFLL )
22     {
23         std[cb0c3ada4437c7f6]::thread::local::panic_access_error();
24     }
25     *v4 = v5 + 1;
26     v6 = v4[1];
27     if ( v6 == 0 )
28     {
29         *v4 = v5;
30         return sub_68CFCE(a1: v2);
31     }
32     v7 = *(__QWORD *) (v4[2] + 64);
33     v8 = sub_617A48(a1: v2);
34     sub_61824A(a1: (v7 == 0) + 1LL, a2: v7, a3: v8);
35     sub_617A57(a1: v2);
36     v9 = sub_6F60A0(a1: *(__QWORD *) (v2 + 56), a2: v6);
37     if ( v9 == 0 )
38     {
39         sub_5D8330(a1: &stru_13C7658);
40         --v4;
41     }
42     v10 = sub_67E50C(a1: &v80, a2: sub_687E03, a3: v9, a4: &v81); // Call the original Math.min
43     if ( v10 != 0 )
44     {
45         sub_6F65A0(a1: &v82, a2: v10);
46         if ( *(int *) (a1 + 16) >= 2 )
47         {
48             v11 = sub_61857A(a1, a2: 1); // Get the second argument (current lap time)
49             v12 = (__int64 *)sub_61821D(a1: v11, a2: v2);
50             if ( v12 != nullptr )
51             {
52                 sub_6F62F0(a1: v12);
53                 sub_67E679(a1: (__int64)v76); //
54                 // Locking access to the global
55                 // vector holding the array of
56                 // the scaled integers
57
58             v13 = v77;
59             if ( v76[0] != 0 )
60             {
61                 goto LABEL_207;
62             }
63             v14 = *(__QWORD *)v77 + 4; // Get the front value of the scaled integers vector
64             if ( v14 != 0 )
65             {
66                 v15 = *(__QWORD *)v77 + 3;
67                 v16 = *(__QWORD *)v77 + 2;
68                 v17 = 0;
69             }
70         }
71     }
72 }

```

Figure 12: The hook for Math.min (Part 1)

```

51 v13 = v77;
52 if ( v76[0] != 0 )
53     goto LABEL_207;
54 v14 = *((_QWORD *)v77 + 4); // Get the front value of the scaled integers vector
55 if ( v14 != 0 )
56 {
57     v15 = *((_QWORD *)v77 + 3);
58     v16 = *((_QWORD *)v77 + 2);
59     v17 = 0;
60     if ( (unsigned __int64)(v15 + 1) >= *((_QWORD *)v77 + 1) )
61     {
62         v17 = *((_QWORD *)v77 + 1);
63         *((_QWORD *)v77 + 3) = v15 - v17 + 1;
64         *((_QWORD *)v13 + 4) = v14 - 1;
65         v79 = *(double *) (v16 + 8 * v15);
66     }
67 sub_69C4A0(a1: v13);
68 if ( v14 != 0 )
69 {
70     sub_5DFF31(a1: (__int64)v76); // Locking access to the global
71                                     // incremental variable holding
72                                     // the current lap index
73 v13 = v77;
74 if ( v76[0] != 0 )
75     goto LABEL_207;
76 v19 = *((_QWORD *)v77 + 1); // Get and increment its value
77 *((_QWORD *)v77 + 1) = v19 + 1;
78 sub_69C4A0(a1: v13);
79 v20 = 0;
80 v21 = -1518639849;
81 for ( i = 0; ; v21 ^= i )
82 {
83     // Hash the current lap index
84     //
85     switch ( v21 )
86     {
87     case -2037941332:
88         v20 = __ROL1__(v20, 3);
89         i = 1408948542;
90         continue;

```

Figure 13: The hook for Math.min (Part 2)

```

143         continue;
144     case 1998798862:
145         i = 775530148;
146         continue;
147     default:
148         break;
149 }
150 if ( v21 == 198409452 )
151     break;
152 }
153 v25 = sub_67F9D8(a1: v19, a2: v20);
154 sub_68003E(a1: v25);
155 v78 = v26; // Calculate shift from hash
156 v73 = v27; // The shift value
157 compiler_builtins::math::full_availability::trunc::trunc(); // Is shift even
158 v29 = (unsigned __int8)(int)fmin(255.0, fmax(0.0, *(double *)&v28)); // Cast the lap time to a single byte
159 // Cast the lap time to a single byte
160 v30 = 0;
161 v31 = -257302408;
162 v83 = v28;
163 while ( 1 )
164 {
165     switch ( v31 )
166     {
167     case -257302408:
168         v30 = -478991323;
169         if ( *(double *)&v28 != (double)v29 )
170             return sub_68CFCE(a1: v2);
171         break;
172     case 333106269:
173         sub_67E633(a1: (__int64)v76); // Locking access to the real flag global variable
174         v32 = v77;
175         if ( v76[0] != 0 )
176         {
177             v13 = v77;
178             goto LABEL_207;
179         }
180         sub_687D50(a1: (_QWORD *)v77 + 1, a2: v29); // Append the lap time as character to
181         // the real flag string
182         sub_69C4A0(a1: v32);
183         v30 = -1563701988;
184         *(_QWORD *)&v28 = v83;
185         break;
186     case -1324285631:
187         v33 = -783651785;
188         v34 = 0;

```

Figure 14: The hook for Math.min (Part 3)

```

496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553

case 1779894813:
    v66 ^= 0x72DF00FF607BE1E2uLL;
    v67 = 902472516;
    break;
case 2053720582:
    v66 ^= 0x31935C3B1D9856uLL;
    v67 = -217412217;
    break;
case -1975610353:
    if ( (v73 & 1) != 0 )
        *(double *)&v28 = -*(double *)&v28; // Flip the sign if the shift is even
    v71 = _mm_sub_pd(
        (__m128d)_mm_unpacklo_epi32(
            (__m128i)((v34 << 7)
                | (32 * v40)
                | (16 * v57)
                | (4 * v63)),
            (__m128i)xmmword_1059F0),
        (__m128d)xmmword_105440); //
//
// Add the shift to the lap time and convert
// the result from degrees to radian
//
v74 = 3.141592653589793
    / (_mm_unpackhi_pd(v71, v71).m128d_f64[0]
        + v71.m128d_f64[0])
    * ((double)(int)v78 + *(double *)&v28);
    compiler_builtins::math::full_availability::round::round(); //
//
// Calculate sine for the result using the
// 16th-degree taylor series expansion
//
v75 = sub_687CFF(
    a1: 0,
    a2: 16 * v66,
    a3: v74 - v72 * 6.283185307179586); //
//
// Locking access to the global
// variable holding the summation
// of the multiplied sine functions
//
sub_604A82(a1: (__int64)v76, a2: dword_1438590);
v13 = v77;
if ( v76[0] == 0 ) //
//
// Multiply the sine result by the scaled integer
// and adding the result to the same global variable
//
*((double *)&v77 + 1) = v79 * v75 + *((double *)&v77 + 1);
sub_69C4A0(a1: v13);
return sub_68CFCE(a1: v2);
default:
    v68 = v66;
    break;
}

```

Figure 15: The hook for Math.min (Part 4)

The Hook for String.prototype.replaceAll

- Calls the original `String.prototype.replaceAll` and stores the result in the return object.
- Compares the summation of scaled integers against the summation of the multiplied sine functions.
- If they match, overwrites the return object with the string built from the lap timings (the real flag) and fixes the character at index 30 from '1' to 'l' by adding 59 to its byte (changing `flare` to `flare`).

```

1  //
2  //
3  // Hook for String.prototype.replaceAll
4  //
5  //
6  __int64 __fastcall sub_67E81E(_QWORD *a1)
7  {
8      // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
9
10     v3 = sub_67E4F2((__int64)a1);
11     v35 = v3;
12     v36 = a1;
13     v25 = sub_6F6550(a1);
14     v4 = sub_618594(a1);
15     v5 = sub_67E50C(a1: &v35, a2: sub_67EB09, a3: v4, a4: &v36); // Call the original String.prototype.replaceAll
16     if ( v5 == 0 )
17         return sub_68CFCE(a1: v3);
18     v6 = v5;
19     sub_6F65A0(a1: &v25, a2: v5);
20     if ( (unsigned __int8)sub_6F6120(a1: v6) == 0 )
21         return sub_68CFCE(a1: v3);
22     v7 = sub_617A48(a1: v3);
23     sub_61818C(a1: &v26, a2: v6, a3: v7);
24     sub_604A82(a1: (__int64)v29, a2: dword_1438590);
25     v8 = v30;
26     if ( (_BYTE)v29 != 0 )
27         goto LABEL_5;
28     v34 = *((unsigned __int64 *)v30 + 1);
29     sub_69C4A0(a1: v30);
30     sub_604A82(a1: (__int64)v29, a2: dword_1438580);
31     v8 = v30;
32     if ( (_BYTE)v29 != 0 )
33         goto LABEL_5;
34     v33 = *((unsigned __int64 *)v30 + 1);
35     sub_69C4A0(a1: v30);
36
37     // Comparing the summation of scaled integers with the summation
38     // of the multiplied sine functions
39     if ( fabs(*(double *)&v34) <= 0.000001
40         || fabs(*(double *)&v33) <= 0.000001
41         || fabs(*(double *)&v34 - *(double *)&v33) >= 0.000001 )
42     {
43         goto LABEL_6;
44     }
45     sub_67E633(a1: (__int64)v29); // Locking access to the real flag global variable
46     v12 = v30;
47     if ( (_BYTE)v29 != 0 )
48     {

```

Figure 16: The hook for String.prototype.replaceAll (Part 1)

```

39  if ( fabs(*(double *)&v34) <= 0.000001
40  || fabs(*(double *)&v33) <= 0.000001
41  || fabs(*(double *)&v34 - *(double *)&v33) >= 0.000001 )
42  {
43      goto LABEL_6;
44  }
45  sub_67E633(a1: (__int64)&v29);           // Locking access to the real flag global variable
46  v12 = v30;
47  if ( (_BYTE)v29 != 0 )
48  {
49      v8 = v30;
50  LABEL_5:
51      sub_69C4A0(a1: v8);
52  LABEL_6:
53      v9 = v26;
54      v10 = v27;
55      goto LABEL_7;
56  }
57  sub_5CC332(a1: v37, a2: *((__QWORD *)&v30 + 2), a3: *((__QWORD *)&v30 + 3));
58  sub_69C4A0(a1: v12);
59  v13 = v26;
60  v14 = v27;
61  v15 = v28;
62  v16 = v37[1];
63  v17 = v37[2];
64  v18 = v37[2];
65  if ( v28 < v37[2] )
66  {
67      v18 = v28;
68  }
69  for ( i = 2138955300; i == -1977242253; i ^= v19 )
70  {
71      v19 = 1766263494;
72      if ( v15 >= 0x1F )
73      {
74          *(_BYTE *)&(v14 + 30) += 59;           // Modify flare to flare
75      }
76  LABEL_29:
77      if ( i == 2138955300 )
78      {
79          for ( j = 0; v17 != j; ++j )
80          {
81              if ( j < v18 )
82              {
83                  *(_BYTE *)&(v14 + j) = *(_BYTE *)&(v16 + j); // Overwrite the decoy flag with the real flag
84              }
85              v19 = -178752681;
86              goto LABEL_29;
87          }
88      }
89  }

```

Figure 17: The hook for String.prototype.replaceAll (Part 2)

The shift generation function at RVA `0x68003E` is an obfuscated implementation of Newton's polynomial interpolation over the points (h_i, Δ_i) , where h_i is the hash of lap index i (the input) and Δ_i is its corresponding shift (the output). This is used instead of hardcoding a shift table in the binary. Since the lap indices and hashing logic are entirely deterministic, we can compute each h_i and evaluate the polynomial directly to extract all 28 shift parameters.

Cracking the Formula

We have the following parameters:

- 28 laps, with each lap index represented by i and the number of laps by N .
- Random scaled integer per lap S_i , with all integers having the same sign.
- Shift per lap Δ_i .
- Lap time (real flag character) t_i .

For the right flag, the following equation must be satisfied:

$$\sum_{i=0}^{N-1} S_i = \sum_{i=0}^{N-1} S_i \cdot \sin(\Delta_i \pm t_i)$$

Since we know that for any real angle θ , the sine function is bounded:

$$\sin(\theta_i) \leq 1 \quad \text{as} \quad \theta_i = \Delta_i \pm t_i$$

And since all scaled integers S_i have the same sign:

$$\sum_{i=0}^{N-1} S_i \cdot \sin(\theta_i) \leq \sum_{i=0}^{N-1} S_i$$

The bound is attained if and only if every individual term simultaneously achieves its maximum:

$$\sum_{i=0}^{N-1} S_i \cdot \sin(\theta_i) = \sum_{i=0}^{N-1} S_i \iff \sin(\theta_i) = 1$$

Thus, the global sum equation decouples into N independent, single-variable equations:

$$\sin(\Delta_i \pm t_i) = 1, \quad t_i \in [0, 255], \quad i \in \{0, \dots, N-1\}$$

In degrees, this is only true when θ is an odd multiple of 90 degrees of the form $90 \cdot (4k + 1)$:

$$\Delta_i \pm t_i = 90 \cdot (4k + 1), \quad k \in \mathbb{Z}$$

That congruence directly yields each real flag character:

$$t_i \equiv \pm(90 - \Delta_i) \pmod{360}$$

Proving that there is only one valid ASCII solution per lap is straightforward given that $t_i \in [0, 255]$.

Getting the Flag

We can extract all 28 shifts dynamically using the IDA debugger:

1. Add a breakpoint at RVA `0x67EEDA`.
2. Run the game until the end of the first lap when the breakpoint is hit.
3. Run the following IDAPython script to collect all 28 shifts and solve for the flag (`File` → `Script file...`):

```
import ida_kernwin, ida_dbg, ida_nalt, ctypes

def get_flag(shifts):
    flag = ""
    for shift in shifts:
        is_even = shift % 2 == 0
        found = False
        for c in range(0, 0xff):
            if (shift + (-c if is_even else c) - 90) % 360:
```

```
        continue

    if found:
        ida_kernwin.warning("Duplicate flag detected")
        return ""

    flag += chr(c)
    found = True

return flag

def main():
    if ida_dbg.get_process_state() >= 0:
        ida_kernwin.warning("The debugger must be running and the process must be suspended")
        return

    rip = get_reg_value("RIP")
    rva = rip - ida_nalt.get_imagebase()
    if rva != 0x67EEDA:
        ida_kernwin.warning("The current RIP must be at RVA 0x67EEDA for the first time")
        return

    lea_va = rip + 0x67EC9F - 0x67EEDA # For resetting RIP to RVA 0x67EC9F
    ida_dbg.add_bpt(rip) # Re-confirm the breakpoint

    signed = lambda x: ctypes.c_int64(x).value
    shifts = [signed(get_reg_value("RAX"))]
    for _ in range(27):
        if rip != get_reg_value("RIP"):
            ida_kernwin.warning("Stopped at unexpected location")
            return

        set_reg_value(lea_va, "RIP")
        ida_dbg.continue_process()
        if ida_dbg.wait_for_next_event(ida_dbg.WFNE_SUSP, 1) == 0:
            ida_kernwin.warning("Waited too long for the breakpoint to hit")
            return

        shifts.append(signed(get_reg_value("RAX")))

    flag = get_flag(shifts)
    if flag:
        ida_kernwin.info(f"The flag is: {flag}@flare-on.com")

if __name__ == "__main__":
    main()
```

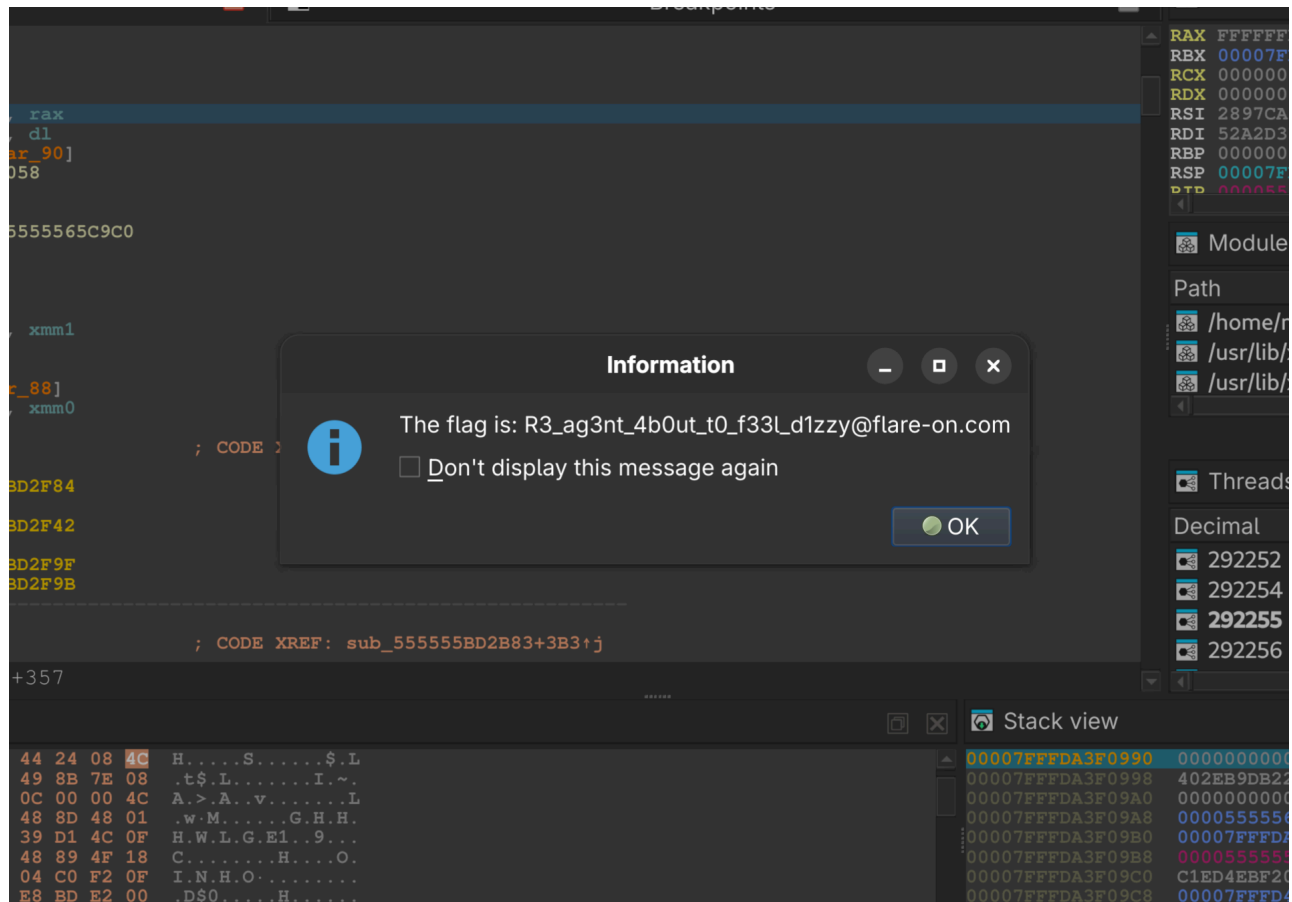


Figure 18: Getting the flag in IDA Pro

The recovered flag is: **R3_ag3nt_4b0ut_t0_f33l_d1zzy@flare-on.com**.

Alternatively, you can replace the lap time calculation line in the backend JS in-memory with the following line so the game displays the real flag upon finishing (after disabling obstacle crashes and increasing car speed):

```
let lapSecs = "R3_ag3nt_4b0ut_t0_f33l_d1zzy"[state.currentLap - 1].charCodeAt(0);
```



Figure 19: Real flag upon finishing with the right lap times

This challenge is inspired by sample

[24e56ef02a783174fb369e9bd1fcbb9fec77a94891b420ed2774a16c572b10dd](#).