「メガ A & B」 Write-up

Vlad Manolescu & Dimitris Tsiplakis

June 30, 2025

1 Abstract of challenge

This challenge consists of a Rust-based game called ”Sonk” (a copy of the Sonic games of the same generation) that was made to simulate and potentially be playable of the Sega Genesis/Mega Drive console. It uses a custom Rust framework called **megarust** that provides low-level hardware abstractions for the Genesis console.

1.1 Problem description

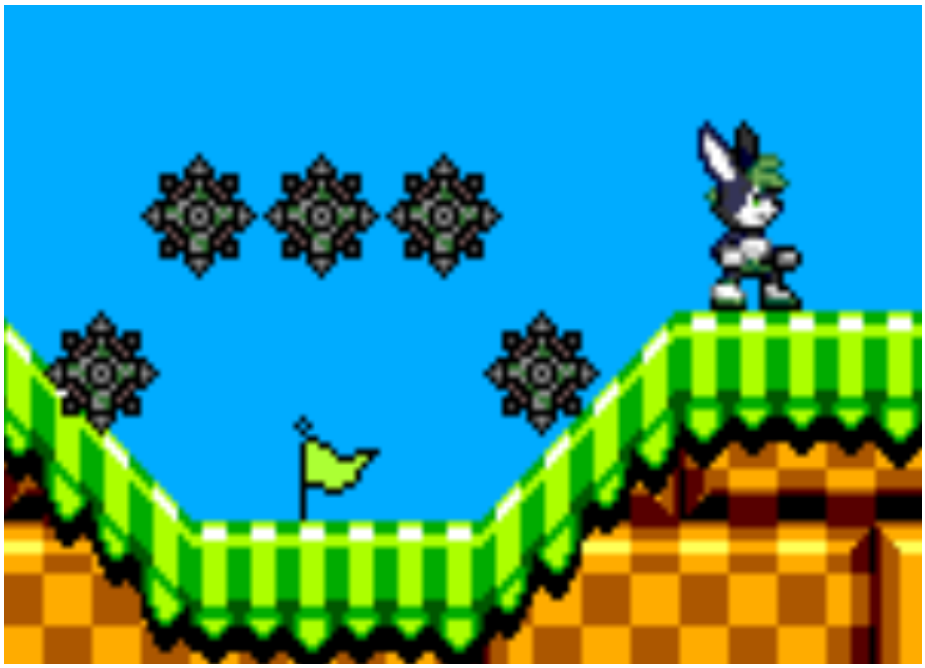
Join the 16-bit revolution! Can you help Sonk the Rabbit find all flags? Submit your recording to get the flag:

`grey{python3 submit.py solution.inp mega.2025.ctfcompetition.com 1337`

Note: Submit the flag starting with ”CTFA:” here. For submitting the other flag, see the challenge titled ”メガB”.

Note 2: To make the メガ challenges less annoying for those who already solved it locally, the ability to get the flag from the server may depend on mame version you installed. This wasn’t obvious and we apologise. The version we run at the server is 0.277+dfsg.1-0ubuntu1 ppa3 noble1 and it comes from ppa:c.falco/mame.

2 メガA (110 solves, solved by Dimitris Tsiplakis)



In this case, we needed to get Sonk between these spikes, without any obvious way of entry. Thankfully, there is a conveniently placed wasp next to this area that, if hit, will launch Sonk backwards from the direction he got hit. This comes into play, because the rendering area of the game is small enough, so that Sonk can be right next to the wasp, while having some of the spikes unloaded from this cluster.

This is where the solution comes into play. In **game.rs**, the function **_update_camera**, responsible with letting the camera keep up with the player, has an vulnerability, where the game unloads the spikes off screen.

```
    for spike: &mut Spike in &mut self.spikes.iter_mut() {
        spike.sprite.x = (spike.sprite.x as i16 - dx) as u16;
        spike.sprite.y = (spike.sprite.y as i16 - dy) as u16;
    }
    self.map.add_new_spikes(dx, dy, &mut self.spikes);
} fn update_camera
```

If the player gets hit by the wasp in an area where some of the spikes are unloaded, Sonk gets pushed outside of the rendered screen, and he can get inside and get the flag.

3 メガ A (87 solves, solved by Vlad Manolescu)

The game contains a critical memory overflow vulnerability in the enemy spawning system that occurs when the camera's render boundary aligns precisely with a wasp spawn point. This causes infinite wasp spawning until the game crashes due to memory exhaustion.

The vulnerability is found inside of **map.rs** in the method called **_add_new_wasps** around line 130

```rust
pub fn add_new_wasps(&self, dx: i16, dy: i16, wasps: &mut Vec<Wasp, N: 256>) {
    for (x: &i16, y: &i16) in map_data::WASPS {
        let x: i16 = *x + self.scroll_x + 128;
        let y: i16 = *y + self.scroll_y + 128;
        if Wasp::off_screen(x: x + dx, y: y + dy) && Wasp::on_screen(x, y) {
            wasps &mut Vec<Wasp, 256>
                .push(item: Wasp::new(x as u16, y as u16)) Result<(), Wasp>
                .map_err(op: |_| "too many waspy bois :C") Result<(), &'static str>
                .unwrap();
        }
    }
}
```

In addition to the previous description, if the player does some constant small camera movements, while the wasp spawn is still on the edge of the render boundary, the spawn condition gets turned on and off constantly, until there are enough wasps that spawn into each other to overflow the game's memory. The game also lacks a duplication prevention system, so the duplicated wasps stay on screen.

Afterwards, the player will glitch up to the top of the screen, and eventually, after running back and forth consistently, the player will get the flag.