Understanding Arbitrary Code Execution

A case study on Pokémon Emerald

Student:
Sasha Toscano

Advisor:
Carlo Alberto Furia

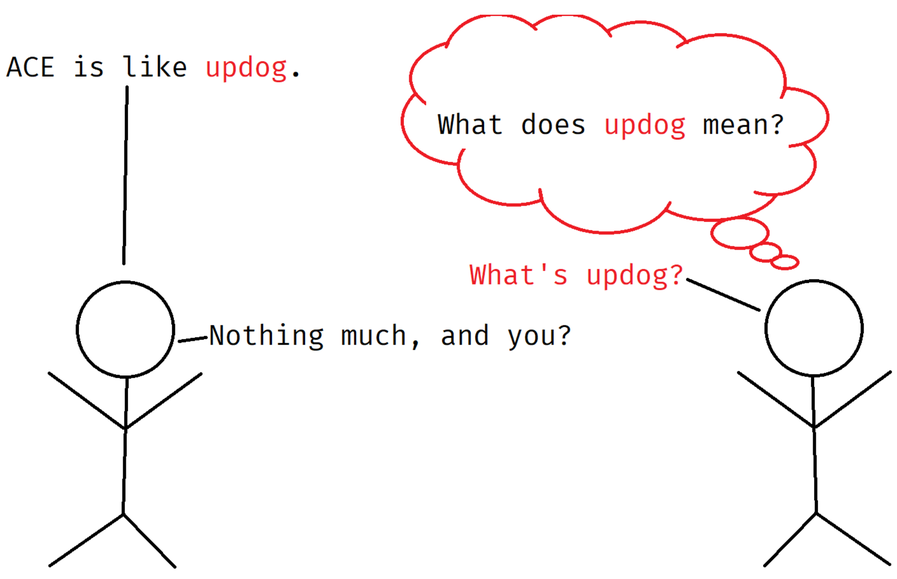
Co-Advisor:
Marc Langheinrich

What is this about?

This project explores how attackers can exploit software bugs to make a program do unintended things through **Arbitrary Code Execution**. It focuses on an example from Pokémon Emerald, a video game from 2004, where players discovered a way to reprogram the game by carefully exploiting in-game glitches.

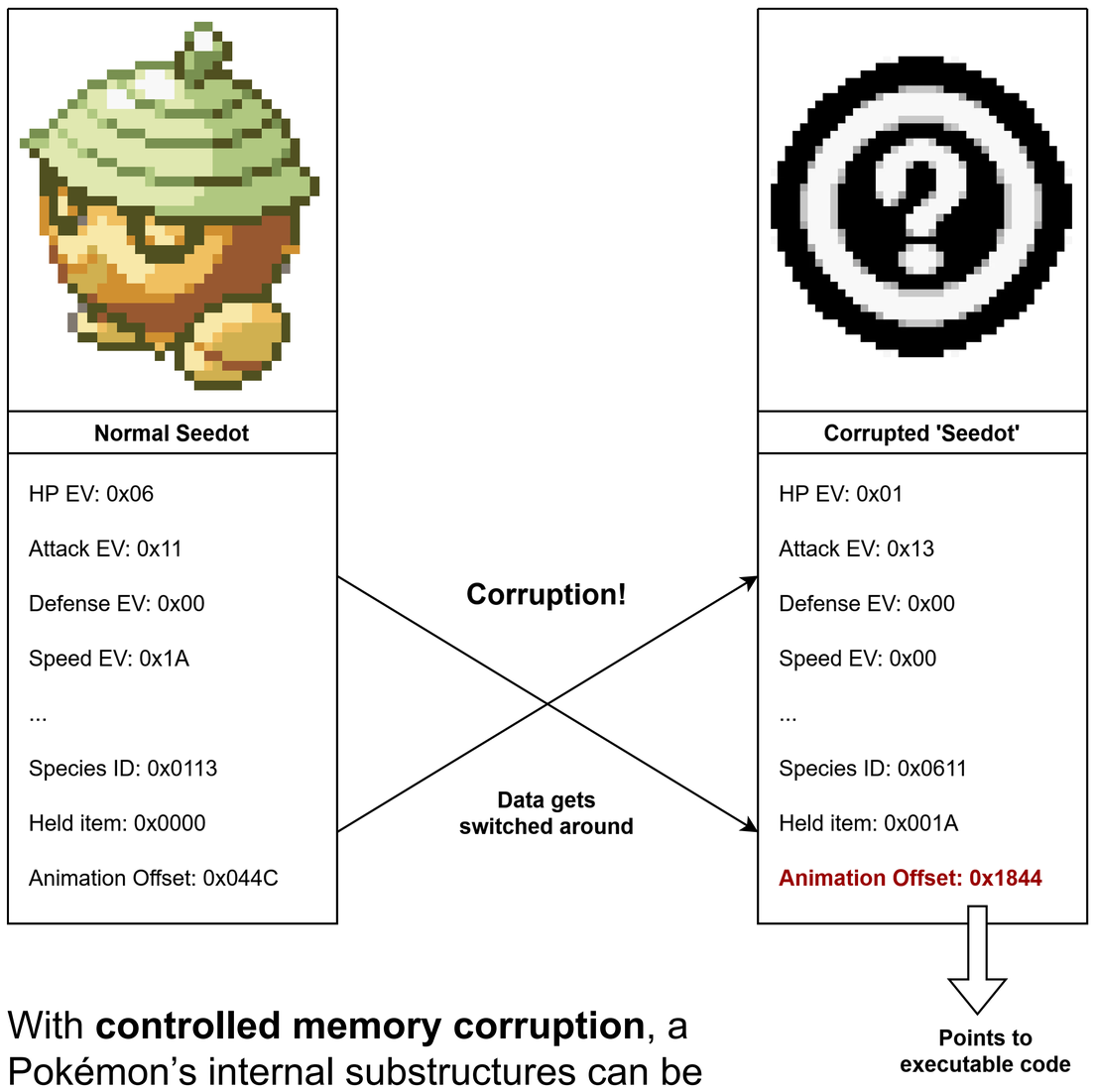
We explain how this works, show how similar attacks happen in other software, and review how modern systems try to detect or prevent such exploits.

What is Arbitrary Code Execution?



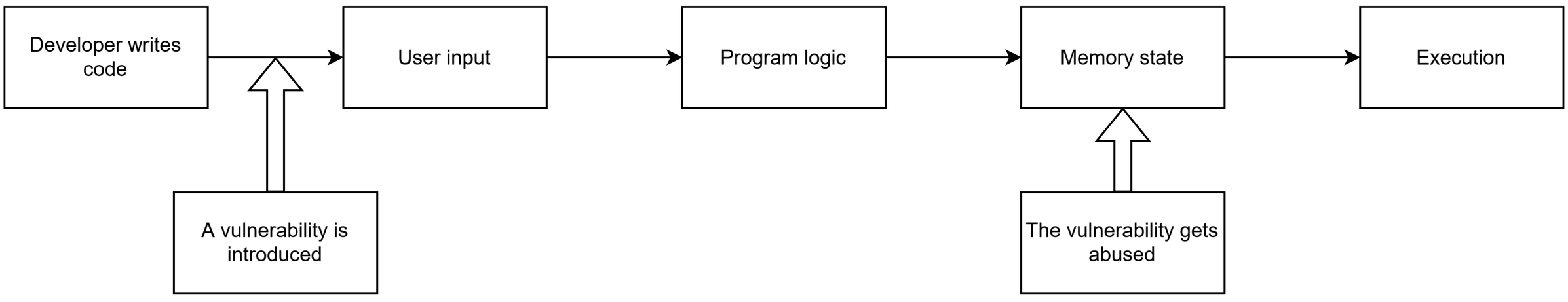
Arbitrary Code Execution (ACE) refers to the ability of an attacker to run unintended code by exploiting software vulnerabilities, potentially leading to **data leaks**, **privilege escalation**, or **full system compromise.**

How does Arbitrary Code Execution happen in *Pokémon Emerald?*



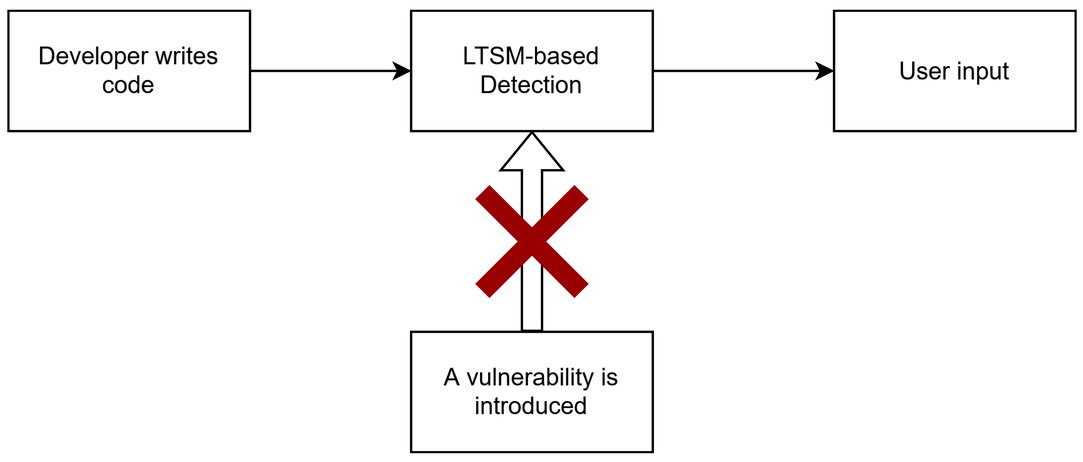
With **controlled memory corruption**, a Pokémon's internal substructures can be reordered. As a result, harmless stats like EVs are **misinterpreted** as a Species ID, and when the game tries to execute something that depends on it, like playing its animation, **the game can end up pointing to user-manipulated data**, resulting in ACE.

Normal exploitable setup



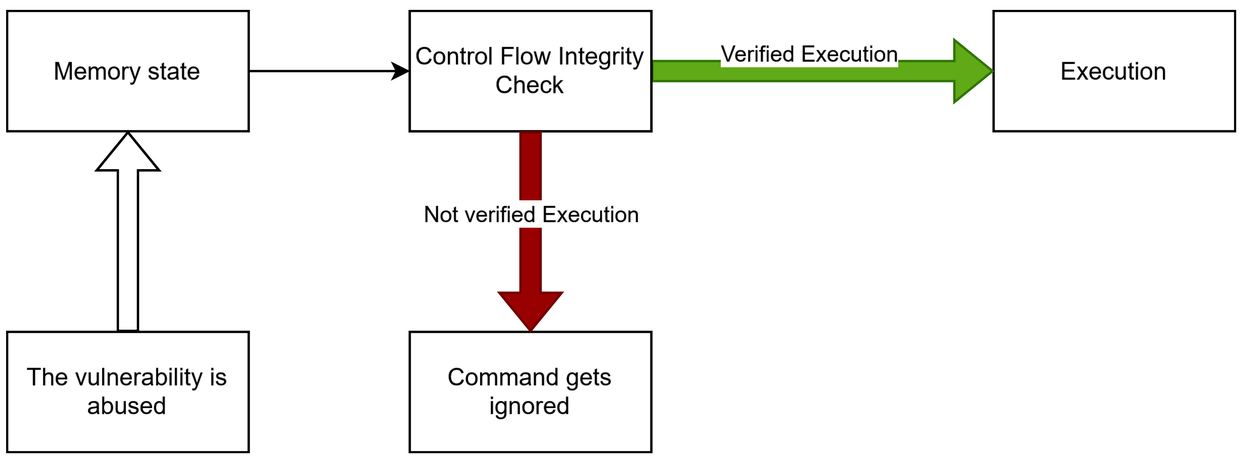
During development, **vulnerabilities** may be introduced by developers. These bugs, if left **undetected**, persist through to runtime. When the program runs, attackers can craft and inject malicious inputs that targets these flaws, altering the memory state and hijacking control of the program. Without **detection** or **prevention** mechanisms, this path enables **arbitrary code execution**.

Detection-based solution



An LSTM model scans source code to detect patterns of exploitable constructs, helping developers fix vulnerabilities before deployment.

Prevention-based solution



G-Free and EC-CFI enforce strict control-flow rules at runtime, preventing hijacked execution even if memory is compromised.