Hack The Box Challenge Proposal: ShinyHunter

Team 6

Introduction

ShinyHunter is an adventure game that simulates a Pokémon game, where the player chooses a Poketmon and starts their journey. This challenge caught our interest because it is a game, and we are familiar with python coding language. The challenge also has a good review score of 4.9 stars and about 412 users have done it.

Challenge Architecture

By analyzing the python code, we determined that the program starts by **generating a random MAC** address by using "random library", then it displays the professor introducing the Poketmon world. The game asks for the user's name and let them choose their starter Poketmon based on three options provided. The game displays the information about the Poketmon chosen. Throughout the game, it also displays "images" of the professor, house, and Poketmons using ASCII. If the Poketmon is shiny, which is **determined by the game by generating a pseudo-random generator**, then the user gets rewarded with a flag. This game is similar to Capture the Flag challenge, which means that the goal of the challenge is to analyze and hack the game to capture the flag.

Possible Vulnerabilities

A simple vulnerability is that inputs are not being checked properly. Attackers could enter commands as input, risking data integrity. Attackers could also enter a long value that could cause a buffer overflow in the memory. Another easily visible vulnerability is that the flag is embedded in the code, making it easy for hackers to easily see the flag. This flag should never be stored within the code.

Another vulnerability is that the **MAC address generation is not fully randomized**. Although they are using the random function, it can still be predictable. The same applies to the **LCG function that generates random seeds,** which are still predictable. An attacker can determine the seed if they obtain the Mac address and the time. Brute Force is another way to obtain the seed.

Lastly, the use of os.system could be a potential vulnerability depending on the environment in which the game is being run. An attacker could enter other commands instead of "clear", putting in hazard the CIA triad in different ways.

Plan to Capture the Flag

Although the code has some vulnerabilities, we cannot exploit them as this program is not connected to a database (SQL injection and Cross Scripting). Therefore, we plan to perform the following considering that we do not have access to the python code:

- **Brute Force the seed generated by LCG based on time and Mac address** until we get a Shiny Poketmon. We'll need to create a program that generates these values. It will take time, so we'll need to look into how long this could take.
- Or Reverse Engineer the process when playing the game.

Game notes

- Predictability of pseudorandom number generator (PRNG)
- Exploring deterministic nature of random module
- Values that are used to compute the shiny value are
    - Trainer ID
    - Secret ID
    - Personal ID
- If the value is less than 8, pokemon is shiny
- Function for pokemon generation is generate_pokemon(seed, …)
- Goal is to determine a seed where the result is shiny pokemon
- Deterministic PRNG – random module is deterministic when initialized with the same seed
- Determinstic means that the future can be calculated exactly without involving randomness
- For a given seed, the sequence of random number (3 values) will always be the same
- Brute-force can be performed to search the possible seeds to get shiny pokemon
- Scripting tools: pwntools
- How does random module work? How does seeding affect the output.
- Brute-force can be optimizd by analyzing the relationship between the seed values and the resulting shiny value

- PRNG: Algorithm that uses mathematical formulas to produce sequences of random numbers. PRNGs are suitable for applications where many random numbers are required and where it is useful that the same sequence can be replayed easily

- Pwntools: CTF framework and exploit development library. Helps to retrieve prompts and send responses to the connection established.

Script notes

- Script uses random.seed() to predict Trainer ID and Secret ID, and iterates Pokemon ID to fiind shiny values (PRNG exploitation)
- Script uses Linear Congruential Generator (LCG) to simulate how the seed may evolve (internal system behavior)
- Initial_seed calculates deterministically the seed using system time and MAC address
- Script gets the MAC address of the connection and uses it to calculate the seed
- Checks all three pokemons to see which one is shiny
- Script delays timing to align with the RNG seed to get the correct value by calculating the time needed to wait
- Script is automated to enter name, choosing pokemon, check if shiny or not and retrieves the flag
- Time sensitive since the game is time based
- Some parameters are hardcoded for LCG constants (a, c, m)
- Pitertools library: Tools to process python iterators in parallel.

- Function to wait for prompts from the server and then send the message. From pwn library,
  - conn.recvuntil - Receive up to and including the string prompt
  - conn.sendline - Send the string s and a newline. (see cheat sheet in references)
- Extract the MAC address by …
- We know their LCG function, we'll use the same values in out LCG function in the script
- We also use the same function for generating ids – trainer and secret
- Generate pokemon function is also about the same, but it only returns whether it is shiny or not.
- Initial seed is not only determined by the current time, but also by the time that has elapsed.

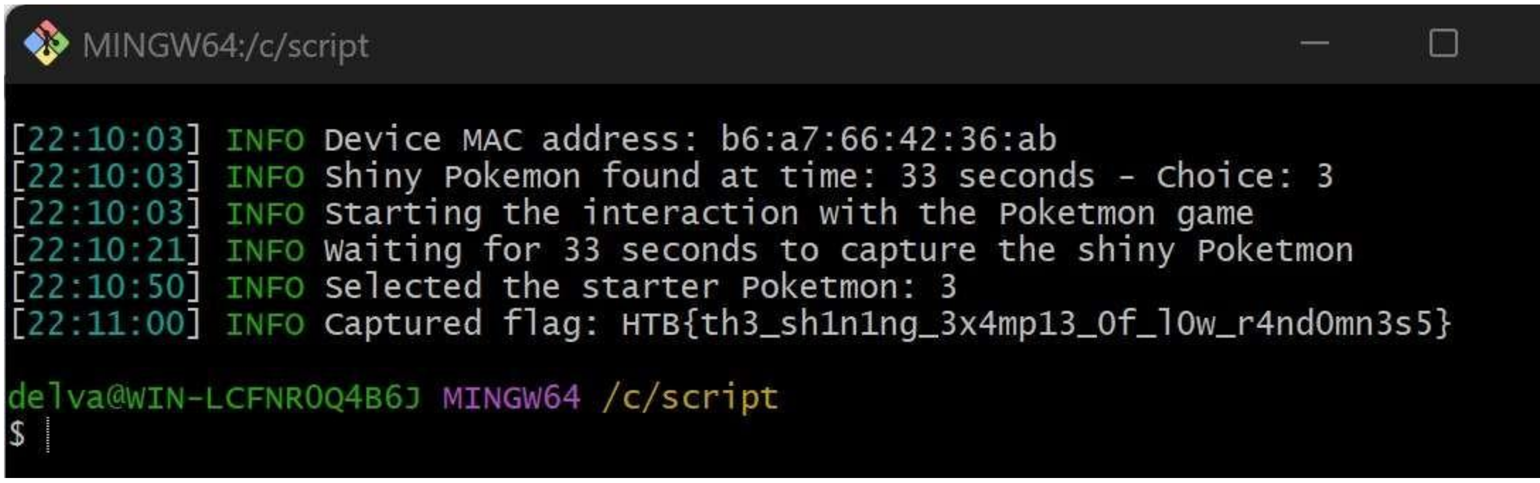
```
    boot_time = time.time()
    system_time = time.time()

    time_passed = time.time() - boot_time
    dialog_time = system_time + time_passed
    formatted_time = int(dialog_time)

    initial_seed = int(formatted_time + int(device_mac.replace(":", ""), 16))
    seed = lcg(initial_seed)
    tid, sid = generate_ids(seed)
```

- Generate_time_to_wait: main function of attack. Finds how many seconds after the system time we need to wait until we get shiny. Tries 3 consecutive seeds (seed, seed+1, seed+2) to simulate 3 starter Pokémon (starter_pokemons[choice]).
- Function to automate the interaction with the game by searching for prompts and sending msgs
- Searching for shiny pokemon function starts with a connection to the server, extracting the MAC address, and generating the time to wait

Improving Script



```
[22:10:03] INFO Device MAC address: b6:a7:66:42:36:ab
[22:10:03] INFO Shiny Pokemon found at time: 33 seconds - Choice: 3
[22:10:03] INFO Starting the interaction with the Poketmon game
[22:10:21] INFO Waiting for 33 seconds to capture the shiny Poketmon
[22:10:50] INFO Selected the starter Poketmon: 3
[22:11:00] INFO Captured flag: HTB{th3_sh1n1ng_3x4mp13_0f_l0w_r4nd0mn3s5}

delva@WIN-LCFNR0Q4B6J MINGW64 /c/script
$
```

- ✓ The second line can be removed
- ✓ Remove the colored terminal
- ✓ Remove loader function
- ✓ Make script ask for host and port
- ✓ Show each try for every attempt (maybe add some wait time between attempts
  - echo "[Attempt #$attempt] $(date '+%Y-%m-%d %H:%M:%S')" | tee -a "$LOG"
  - Output to TXT?
- ✓ Show the values of shiny Pokémon
- ✓ Ideas to modify the logic for brute force optimization
- ✓ ...

Code notes

- Libraries used: os, time, random, uuid, sys
- Some important functions in the game code:

```python
def get_mac():
    mac = [random.randint(0x00, 0xff) for _ in range(6)]
    return ":".join("{:02x}".format(x) for x in mac)

def lcg(seed, a=1664525, c=1013904223, m=2**32):
    return (a * seed + c) % m

def generate_ids(seed):
    random.seed(seed)
    tid = random.randint(0, 65535)
    sid = random.randint(0, 65535)
    return tid, sid

def generate_poketmon(seed, tid, sid, name):
    random.seed(seed)
    stats = {
        "HP": random.randint(20, 31),
        "Attack": random.randint(20, 31),
        "Defense": random.randint(20, 31),
        "Speed": random.randint(20, 31),
        "Special Attack": random.randint(20, 31),
        "Special Defense": random.randint(20, 31)
    }
    natures = ["Adamant", "Bashful", "Bold", "Brave", "Calm", "Careful", "Docile", "Gentle",
"Hardy", "Hasty", "Impish", "Jolly", "Lax", "Lonely", "Mild", "Modest", "Naive", "Naughty",
"Quiet", "Quirky", "Rash", "Relaxed", "Sassy", "Serious", "Timid"]
    nature = random.choice(natures)
    pid = random.randint(0, 2**32 - 1)
    shiny_value = ((tid ^ sid) ^ (pid & 0xFFFF) ^ (pid >> 16))
    is_shiny = shiny_value < 8

    return {
        "name": name,
        "stats": stats,
        "nature": nature,
        "is_shiny": is_shiny
    }

    initial_seed = int(formatted_time + int(device_mac.replace(":", ""), 16))
    seed = lcg(initial_seed)
    tid, sid = generate_ids(seed)
```

Random module:

For integers, there is uniform selection from a range.

random.**randint**(*a*, *b*)

Return a random integer *N* such that a <= N <= b. Alias for randrange(a, b+1).

- Produces integers in a closed interval [a, b]
- Uniform probability. Probability 1/(b-a+1)

random.**seed**(*a=None*, *version*=2)

Initialize the random number generator.

If *a* is omitted or None, the current system time is used. If randomness sources are provided by the operating system, they are used instead of the system time (see the os.urandom() function for details on availability).

If *a* is an int, it is used directly.

With version 2 (the default), a str, bytes, or bytearray object gets converted to an int and all of its bits are used.

- Uses Mersenne Twister PRNG

random.**choice**(*seq*)

Return a random element from the non-empty sequence *seq*. If *seq* is empty, raises IndexError.

- Uniform distribution as well with probability 1/len(seq) -1)

- LCG is a math formula for generating a sequence of pseudorandom numbers. Xn= current seed, a= multiplier, c = increment, m= modulus, Xn+1=next seed. The values are hardcoded to the code because they are commonly used as they produce a long pseudo-random sequence, by maximizing the period and uniformity of the output.

$$X_{n+1} = (aX_n + c) \mod m$$

- Initial seed in the game is determined by using the time and mac address of the machine. Seed is generated using the LCG function, let passed to the other functions to generate values.