Reto - CTF

Monreal De la Rosa, Diego Azael
343427

Quistian Navarro, Juan Luis
341807

Ing. Sistemas Inteligentes, Gen. 2021
Principios de Seguridad Informática

28 de septiembre de 2024

Resumen

Este documento presenta las soluciones a los desafíos de Capture The Flag (CTF) Coding100, Coding200, Coding300 y Coding500. Se describen las soluciones implementadas para resolver cada reto, que incluyen búsqueda de palabras en una cuadrícula, exploración de rutas mediante búsqueda en anchura (BFS), restauración de imágenes esteganografiadas y desarrollo de un intérprete para un lenguaje específico. Cada desafío culmina en la obtención de una bandera. Las soluciones completas están disponibles en un repositorio de GitHub.

Figura 1: Score acumulado de los 4 desafios

Coding 100

Para encontrar las palabras, para cada una se checa cada celda de la cuadricula. Si coincide con la primera letra de la palabra, empieza un proceso de búsqueda en todas las direcciones. Si la dirección no es valida, queda fuera de la cuadricula, o la letra no coincide con la siguiente de la palabra se termina esa búsqueda. Para el patrón de L, si no se ha echo un cambio de dirección y la dirección actual no es diagonal, se checa realiza la búsqueda también en las direcciones perpendiculares. Si se llega al final de la palabra, se marca cada posición recorrida. Una vez hecho para cada palabra, se eliminan las celdas marcadas y se obtiene la contraseña.

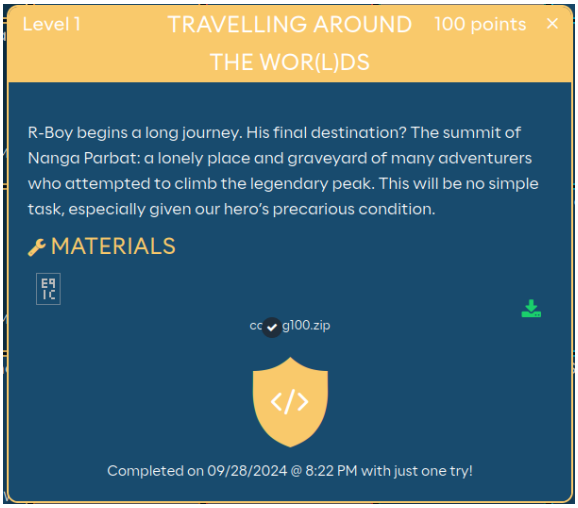

Figura 2: flag: {FLG:c0n9r4t5_3_3nj0y_th3_4dv3ntur3}

Coding 200

Para resolver este problema, se implementó una búsqueda en anchura (BFS) para explorar las rutas desde el inicio hasta el final. Las estrategias clave fueron:

1. **Cargar el mapa**: Se identificaron las posiciones de inicio ('A'), final ('B') y portales, almacenando sus posiciones gemelas.
2. **Simulación del mapa**: En cada paso, las celdas se actualizaban dinámicamente según el número de agujeros negros a su alrededor, afectando también a los portales.
3. **Portales y teletransporte**: Los portales permitían teletransportarse a su gemelo instantáneamente, pero se desactivaban tras su uso, por lo que había que calcular su uso óptimo.

4. **Optimización de ruta**: Se priorizaron las rutas más cortas y aquellas que utilizaban más portales, ordenándolas lexicográficamente en caso de empate.

5. **Salida**: La contraseña para la solución final tenía el formato N-s-P, donde N es el número de rutas óptimas, s es la concatenación ordenada de las rutas, y P el número de portales usados.

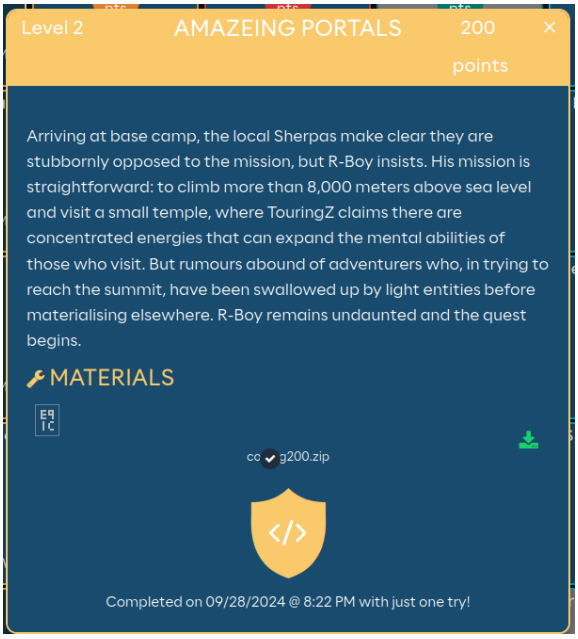

Figura 3: flag: {FLG:4v0Id1N6_bL4Ck_h0l35_c4N_b3_7r1cKy}

Coding 300

Para este desafío, se implementó un script que sigue los siguientes puntos clave:

1. **Cargar la imagen inicial**: El programa lee una imagen que contiene un mensaje esteganografiado. Este mensaje se descifra utilizando la librería stepic para obtener una semilla (seed).

2. **Desordenar y restaurar píxeles**: Con la semilla obtenida, se utiliza numpy para reproducir el desorden de píxeles original y luego invertirlo, restaurando la imagen correcta.

3. **Rotación de sub-bloques**: La imagen se divide en 16 bloques. Cada bloque contiene un mensaje binario esteganografiado, que indica cuánto ha sido rotado el bloque y su posición original. Los bloques se reorganizan y rotan a su estado correcto.

4. **Reconstruir un puzzle Sudoku**: Una vez restaurada la imagen, se extraen los números que conforman un puzzle Sudoku de 16x16. Estos números se obtienen usando OCR (reconocimiento óptico de caracteres) mediante pytesseract.
5. **Resolver el Sudoku**: Con los números extraídos, se resuelve el puzzle utilizando la librería Sudoku. La solución se concatena para formar una contraseña.
6. **Descomprimir archivos**: La contraseña se usa para extraer el siguiente archivo ZIP, continuando con el siguiente nivel del juego.

Este proceso se repite hasta que no se puedan extraer más niveles.

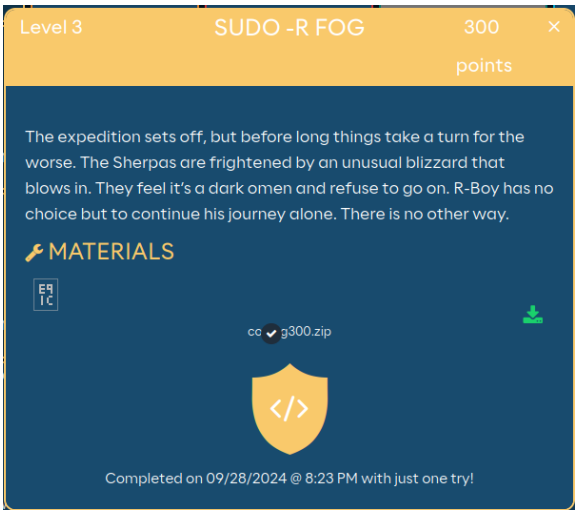

Figura 4: Flag: {FLG:rE9EnEr4T3_My_me5sy_sud0Ku}

Coding 500

Para poder hacer el interprete, primero se analizaron los ejemplos para determinar el significado de cada expresión.

- El lenguaje obtenido fue el siguiente:
- Las expresiones son leídas de izquierda a derecha.
- Los objetos «STRINGS» comienzan con «B», con cada carácter separado por una 'b', y en orden inverso.

- Los objetos «INTEGERS» comienzan con «N» y pueden formarse por una operación aritmética, donde cada elemento se separa por un operador: la 'a' es suma, la 'm' es multiplicación, la 'd' es división, y la 's' es resta.
- La operación «PRINT» se indica con una 'P', y se imprime el valor de la siguiente expresión.
- Las variables se declaran con una 'V', seguido de un nombre de variable, con cada carácter separado por una 'r' y en orden inverso. Si es precedida por un '=', se está asignando el valor, si es cualquier otra expresión, se está evaluando o leyendo.
- Las operaciones 'ADD', 'MULT', 'DIV', 'SUB' siempre son seguidas por dos expresiones, y se evalúan de izquierda a derecha.

El interprete comienza haciendo un split de la entrada para separar cada expresión, para esto se uso una expresión regular que separara en los cambios a mayúscula, en los espacios, y en las operaciones. Para cada línea se invierte el orden, y se insertan a una pila.

Una vez se separan, empieza un ciclo que se repite hasta que la pila esté vacía. En cada iteración se saca un elemento de la pila, si es una operación, se sacan los dos elementos siguientes y se realiza la operación, se guarda el resultado en la pila. Si es una variable, se guarda en un diccionario con el nombre de la variable como llave. Si es un print, se imprime el siguiente elemento.

Para las condiciones, se guarda cada condición individual en una pila y se evalúa de izquierda a derecha. Después se separan las instrucciones para el caso verdadero y la condición alternativa, si es que hay una, en un pila diferente. Una vez separadas, si se cumplió la condición, se inicia un ciclo de ejecución de la pila correspondiente, y sucede lo mismo si se cumple la condición alterna. Las instrucciones pueden tener condiciones anidadas, por lo que al separar ambas ramas, se lleva un contador para saber en cual nivel de anidación se encuentra.

Los bucles funcionan de una forma similar, se guarda la condición en una pila, y se separan las instrucciones a ejecutar. Se crea un copia de la pila de condición y se evalúa, de ser verdadera, se ejecuta el ciclo, haciendo una copia de las instrucciones a ejecutar, y se repite el proceso hasta que la condición sea falsa. Al también haber anidación, también se lleva un contador para saber en que nivel de anidación se encuentra.

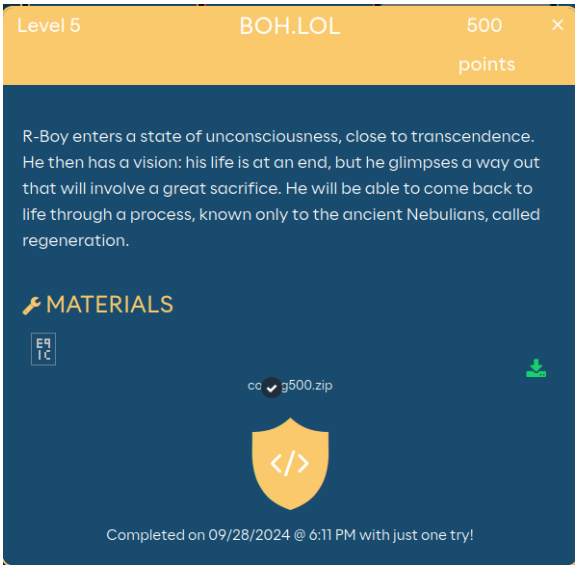

Figura 5: Flag: {FLG:50M371m3Z_350l4Ng_c4N_B3_M0r3_R34d4bL3_7h4N_1337_Fl49z}

Soluciones en github

Las soluciones fueron subidas a github, se pueden consultar en el siguiente enlace: `https://github.com/juanQNav/Capture_the_flag`