TP2 - Bomba binaria

Ilan Nomberg
inomberg@udesa.edu.ar

Contents

Chapter 1

Las Fases

1.1 Fase 1

La fase 1 consistió en una verificación directa de una cadena de texto. El análisis del ensamblado mostró que se realiza una comparación entre el input del usuario y una cadena constante. A continuación se detallan los pasos seguidos:

1. **Análisis del ensamblado:** En la función `phase_1`, se detectó la siguiente secuencia de instrucciones:
   - Carga en `rsi` una dirección constante correspondiente a una cadena ubicada en la sección `.rodata`.
   - Llama a la función `strings_not_equal`, que compara el input con esa cadena.
   - Si los strings no coinciden, se llama a `explode_bomb`.
2. **Determinación de la cadena esperada:** Se identificó la dirección `0x4c9a60` como el lugar donde reside la cadena constante. Utilizando GDB o `objdump -s`, se accedió a esa dirección y se obtuvo el string esperado.
3. **Resultado:** La cadena que permite desactivar esta fase es:

   `He conocido, aunque tarde, sin haberme arrepentido, que es pecado cometido el decir ciertas verdades`

4. **Validación:** Si el input ingresado coincide exactamente (incluyendo puntuación y espacios) con esta cadena, la fase se desactiva con éxito.

1.2 Fase 2

La fase 2 requiere que el usuario ingrese una línea con tres números enteros separados por espacios. El análisis del ensamblador reveló que el programa realiza operaciones bit a bit sobre estos valores y luego llama a una función auxiliar denominada `misterio`.

1. **Lectura del input:** Se parsean tres valores enteros usando la función `__strtol`:

   $$\mathtt{x},\ \mathtt{y},\ \mathtt{z}$$

2. **Verificación intermedia:** Se calcula:

   $$(\mathtt{y} \oplus \mathtt{x}) \gg 1$$

   y se compara con `z`. Si el resultado no coincide, la bomba explota.
3. **Función `misterio`:** Luego de la comparación, se llama a `misterio(z)`, que explota la bomba si el valor es mayor o igual a cero. El análisis en ensamblador mostró que:
   - Si `z` es negativo, la función retorna normalmente.

- Si `z` es cero o positivo, se llama a `explode_bomb`.

4. **Condiciones necesarias para desactivar la fase:**
   - $(\texttt{y} \oplus \texttt{x}) \gg 1 = \texttt{z}$
   - $\texttt{z} < 0$
5. **Ejemplo de input válido:**

   ```
   -5 6 -2
   ```

   Justificación:
   - $-5 \oplus 6 = -3$, y $-3 \gg 1 = -2$
   - $z = -2$ cumple la condición de igualdad y además es negativo

1.3 Fase 3

La fase 3 requiere que el usuario ingrese una línea compuesta por una palabra y un número entero, en ese orden. El análisis del ensamblador mostró que se realiza una búsqueda binaria recursiva sobre un arreglo de strings, y que se acumulan los índices visitados durante la búsqueda para compararlos con el número ingresado. La lógica se implementa mediante una función llamada `cuenta`.

1. **Lectura del input:** El input es procesado mediante `sscanf` con el formato `"%s %d"`, lo que indica que primero se espera una `palabra` y luego un `número` entero `num`. Además, se carga un arreglo de strings (ordenado alfabéticamente) mediante la función `readlines`.
2. **Función `cuenta`:** Esta función implementa una búsqueda binaria recursiva que compara la palabra ingresada con las del arreglo. En cada llamada recursiva se calcula el índice central `mid` usando el siguiente patrón libre de overflow:

   $$\texttt{mid} = (\texttt{low} \oplus \texttt{high}) \gg 1 + (\texttt{low}\&\texttt{high})$$

   En cada paso, si no se encuentra la palabra, se continúa la búsqueda a la izquierda o derecha del arreglo, según el resultado de la comparación.
3. **Acumulación de índices:** Cada vez que la función `cuenta` llama recursivamente a sí misma, acumula el índice `mid` correspondiente. El valor final retornado por la función es la suma de todos los índices visitados durante la búsqueda binaria.
4. **Verificación final:** Luego de ejecutar `cuenta`, el programa realiza las siguientes verificaciones:
   - El valor retornado debe coincidir exactamente con el número `num` ingresado por el usuario.
   - El valor también debe ser estrictamente mayor que 9999 (`0x270f`).

   Si alguna de estas condiciones no se cumple, la bomba explota.
5. **Ejemplo de input válido:**

   ```
   ababillarse 10767
   ```

   Justificación:
   - La palabra `ababillarse` se encuentra en el arreglo de líneas utilizado internamente.
   - La búsqueda binaria recursiva acumula los índices visitados, dando como resultado la suma `10767`.
   - Este valor coincide con el número ingresado y es mayor que 9999, por lo que la fase se desactiva correctamente.

1.4 Fase 4

La fase 4 de la bomba consistió en una función que procesa una cadena de exactamente seis caracteres, aplica una transformación byte a byte mediante una tabla de sustitución, y compara el resultado contra una cadena esperada. A continuación se detallan los pasos seguidos para resolverla:

1. **Análisis del ensamblado:** Se utilizó `objdump` y GDB para analizar la función `phase_4`. Se observó que la función:
   - Lee un string del input y verifica que su longitud sea exactamente 6.
   - Aplica una transformación a cada caracter: realiza `char & 0x0F` y usa ese valor como índice en una tabla de 16 bytes.
   - Compara el string resultante con un string secreto utilizando `strcmp`.
2. **Extracción de la tabla de transformación:** Se identificó la dirección de la tabla `array.0` en la sección `.rodata`, específicamente en `0x4cde50`. Se extrajeron sus 16 bytes utilizando `objdump` o `gdb`, resultando en la siguiente tabla:

   `['e', 'g', 'm', 'c', 'f', 'a', 'i', 'j', 'o', 'p', 'n', 'h', 'd', 'b', 'k', 'l']`

3. **Detección del string esperado:** Se colocó un breakpoint en GDB justo antes de la llamada a `strcmp`, y se inspeccionó el contenido del registro `$rsi` con el comando `x/s $rsi`. El string esperado fue:

   `"lechon"`

4. **Inversión de la tabla:** Se construyó una versión invertida de la tabla para encontrar todos los caracteres ASCII imprimibles que satisfacen la condición `char & 0x0F == índice_requerido`. Esto permitió generar todas las combinaciones posibles de strings de longitud 6 que se transformaran en `"lechon"`.
5. **Generación del input válido:** Se desarrolló un script en Python que probaba todas las combinaciones posibles, generando strings candidatos como por ejemplo:

   `opskHZ`

6. **Validación final:** Se verificó que:
   - El input tuviera exactamente 6 caracteres.
   - La transformación resultara en `"lechon"`.
   - El input fuera ingresado correctamente, incluyendo un `Enter` final que marcara el `EOF`.

   Como resultado, la fase 4 fue desactivada con éxito al ingresar uno de los strings válidos generados.

1.5 Fase Secreta

Luego de desactivar las primeras tres fases, es posible activar una **fase secreta** oculta en el binario, accesible únicamente si se proporciona una entrada especial en la tercera línea del input.

Activación de la fase secreta

La función `phase_defused` contiene una verificación especial que se ejecuta al finalizar la tercera fase:

- Si se han desactivado exactamente tres fases (`num_input_strings == 4`),
- Se evalúa si el tercer input es parseable con el formato `%s %d %s`.
- Luego, se compara el tercer string con la clave secreta: `abrete_sesamo`.

  Si esta condición se cumple, se ejecuta la función `secret_phase()`.

Lógica de `secret_phase`

La fase secreta solicita un número entero entre 1 y 1001. Internamente, esta entrada se utiliza en una búsqueda binaria dentro de un árbol predefinido, con raíz en la dirección `0x4f91f0`.

La función `fun7(node, valor)` recorre el árbol binario comparando el número ingresado con los valores almacenados en los nodos. Esta función retorna:

- `0` si el valor se encuentra exactamente en el nodo raíz,
- Un valor distinto de cero si el número está en otro nodo (según el camino codificado como un entero),
- `-1` si el número no está presente en el árbol.

El programa explota si el resultado de `fun7` es distinto de cero.

Resolución

La raíz del árbol contiene el valor `36` (obtenido inspeccionando la memoria en GDB). Por lo tanto, el input válido para pasar esta fase es:

```
36
```

Input completo de ejemplo

```
He conocido, aunque tarde, sin haberme arrepentido, que es pecado cometido el decir ciertas verdades
-5 6 -2
ababillarse 10767 abrete_sesamo
opskHZ
36
```

- Las dos primeras líneas desactivan las fases 1 y 2.
- La tercera línea desactiva la fase 3 y activa la fase secreta mediante la palabra clave `abrete_sesamo`.
- La cuarta línea desactiva la fase 4 (los primeros seis caracteres se transforman en `lechon`).
- La quinta línea contiene el número correcto para desactivar la fase secreta.

Con esta entrada, la bomba finaliza exitosamente mostrando el mensaje oculto de la fase secreta.