

XSoldier & XKobo – Facciamo il cheat a questi 2 giochi

Pubblicato da **bIAAd!** il 09/02/2001

Livello **base**

Introduzione

Il target di questo tute è alquanto anomalo. Voglio infatti mostrare degli esempi di 'cheating' a dei giochi inclusi e liberamente distribuiti su versioni di Linux quali Mandrake.

Si tratta di semplici 'spara e fuggi' in 2D, tutto sommato difficilini. Per risolverli ci aiuteremo, modificando i programmi in maniera tale da renderci 'immortali'.

Analizziamo i giochi XSoldier ed XKobo, ed in particolare la loro parte di codice relativa al conteggio delle 'vite'.

Programmi usati

- **vim**: editor di testo visuale. La versione GUI del VI di konsole.
- **dasm**: script perl per disassemblare sotto Linux.
- **biew**: editor esadecimale

Iniziamo

Bene, cominciamo con XSoldier. Se avete provato a giocare, vi sarete accorti che dopo i primi 2 o 3 livelli, il gioco comincia a diventare estremamente difficile. Allora proviamo a modificare il programma come detto sopra. Ma come?

Copiamo il gioco in una directory che più ci fa comodo, e facciamo partire il dasm scrivendo:

```
"/percorso/dasm.pl /percorso/xsoldier /percorso/xsoldier.out"
```

Ok, ora editiamo il file xsoldier.out con VIM (la versione 'grafica' di -vi- per konsole), ed analizziamo la dead-list. Qui purtroppo non troveremo nulla di utile, nessun riferimento esplicito a vite, alieni, punteggio... Cosa facciamo allora? Se vi ricordate, alla fine della partita avevamo il classico 'GameOver'.

Quindi, sempre col nostro editor, cerchiamo la stringa 'Game Over'. Questo ci porterà a:

```
0x0804aad0 mov 0x80559a8,%eax
0x0804aae1 cmpl $0x0,0x134(%eax)
0x0804aae8 jne 0x0804ab18
0x0804aaea add $0xffffffff,%esp
0x0804aaed mov 0x8055aac,%eax
0x0804aaf2 push $0x9
```

Possible reference to string: "Game Over"

```
0x0804aaf4 push $0x8052d00
0x0804aaf9 push $0x12c
0x0804aafe push $0xe6
0x0804ab03 push %eax
0x0804ab04 mov 0x80559d0,%eax
0x0804ab09 push %eax
0x0804ab0a mov 0x8055988,%eax
0x0804ab0f push %eax
```

Reference to function : XDrawString

```
0x0804ab10 call 0x08048e58
```

Cerchiamo di capire cosa succede nelle poche righe sopra. In 0x0804aae1 c'è un cmpl ed un seguente jne che insospettiscono molto. Se il salto viene eseguito, il programma prosegue normalmente, in caso contrario viene caricata la stringa 'Game Over', e stampata tramite la funzione di X-Window : XDrawString.

È ovvio che 0x134(%eax), conterrà il numero di vite a disposizione. Se questo numero diventasse 0, si avrebbe il 'Game Over'. È allora intuitivo sostituire al jne un jmp. Fermandoci a questo però, si continuerebbe a modificare il contatore di vite, rischiando un 'collasso' in altre parti del programma. Per evitare il reversing completo del gioco, rendiamo allora, la patch più sicura, e capace di reincrementare il contatore di vite, nella seguente maniera:

```
0x0804aad0 mov 0x80559a8,%eax
0x0804aae1 incl 0x134(%eax)
0x0804aae7 nop
0x0804aae8 jmp 0x0804ab18
```

Per implementare ciò, basterà sostituire i bytes con BIEW, a partire da 0x0804aae1 con:

```
ff, 80, 34, 01, 00, 00, 90, eb
```

E per ciò che riguarda XSoldier è tutto. Potete comunque cercare di fare altre modifiche al programma, come il potenziamento automatico delle armi, o annullare le collisioni della vostra astronave, ecc..

Passiamo ora al secondo gioco, XKobo. Qui in realtà il cheat è già previsto dal programma, caricandolo da console con lo switch '-cheat'. Solo che vogliamo trovare la nostra 'gabola' e quindi ce ne freghiamo....). Cerchiamo anche qui un sistema che ci permetta di giocare con vite infinite.

Disassembliamo al solito il programma con il Dasm, ed editiamolo con VIM. Ora all'interno dell'editor, cerchiamo la stringa 'SHIPS', che nel gioco segna il numero di vite. Ci troveremo nella parte di codice:

```
Referenced from jump at 080514f4 ; 080514f9 ; 080514fe ; 08051509 ;  
Possible reference to string:
```

```
"SHIPS"  
0x08051527 sub $0x8054809,%eax  
0x0805152c push %eax
```

```
Possible reference to string:  
"SHIPS"
```

```
0x0805152d push $0x8054809  
0x08051532 mov 0x806ee9c,%eax  
0x08051537 push $0x55  
0x08051539 push %esi  
0x0805153a push %eax  
0x0805153b mov 0x806ed60,%eax  
0x08051540 push %eax  
0x08051541 mov 0x8061440,%eax  
0x08051546 push %eax
```

```
Reference to function : XDrawImageString
```

```
0x08051547 call 0x080491dc  
0x0805154c add $0x20,%esp  
0x0805154f cmpl $0x0,0x8075310  
0x08051556 jne 0x08051568  
0x08051558 add $0xffffffffc,%esp  
0x0805155b mov 0x808d5a8,%eax  
0x08051560 push %eax
```

```
Possible reference to string:  
"%09d"
```

```
0x08051561 push $0x80547fe  
0x08051566 jmp 0x08051570
```

```
Referenced from jump at 08051556 ;  
0x08051568 add $0xfffffffff8,%esp
```

```
Possible reference to string:  
"999999999"
```

```
0x0805156b push $0x805480f
```

```
Referenced from jump at 08051566 ;  
0x08051570 push %ebx
```

```
Reference to function : sprintf  
0x08051571 call 0x080495fc
```

Questa parte di codice individua lo stato del programma. Se abbiamo scelto l'opzione '-cheat', ci troveremo con '999999999' vite, altrimenti (come nel nostro caso), avremo per il gioco 'normale', le nostre misere 5 'SHIPS'. Questo è realizzato all'indirizzo 0x08051556 con un jne che in base al cmpl immediatamente precedente determina le condizioni di gioco.

Sotto il jne, vediamo subito che in eax viene copiato il valore contenuto in 0x808d5a8, e sotto c'è il 'push' alla stringa "%09d", ed infine un jmp, a cui fa seguito a sua volta una call alla funzione 'sprintf'. Cosa ci suggerisce tutto ciò?

Semplice, che il codice originale era del tipo: sprintf("SHIPS %09d",ships), per la visualizzazione del numero di astronavi a disposizione, contenuto nella variabile ships(rinominata così da me, a sostituzione dell'indirizzo 0x808d5a8), come numero decimale a 9 cifre.

Cerchiamo allora, ammesso che la nostra (del resto palese) supposizione sia corretta, dove viene 'trattato' l'indirizzo 0x808d5a8, operando un semplice Find, sempre all'interno del nostro editor VIM, per la stringa "0x808d5a8". Fatto? Dopo un paio di 'Find

Next' finiremo all'interno della parte di codice:

```
0x08051eda decl 0x808d5a8
```

```
Referenced from jump at 08051ed8 ;  
0x08051ee0 cmpl $0x0,0x808d5a8  
0x08051ee7 jg 0x08051f44
```

Avrete già capito cosa accade qui. In caso di collisione il contatore ships viene decrementato. Se non è ancora giunto a 0, il gioco continua normalmente, altrimenti termina... Per risolvere il problema basterà nappare l'istruzione decl, inserendo 6 bytes col valore 0x90, a partire dall'indirizzo 0x08051eda, ed il gioco è fatto (capita la battuta??).

Credo sia tutto per questo tutorial, quindi vi auguro buon divertimento con i programmi sopra... e un felice 'Linux Reversing';)
byz by blAAAd!
-eLeNg-

Conclusioni

Un saluto a tutti quelli della RACL ovviamente. Il tutorial come avete visto, non era particolarmente difficile. Speriamo solo, che il target 'incuriosisca' qualcuno.

Ah già, dimenticavo... visitate il mio sito: MASPAREV.CJB.NET (scusate la pubblicità poco occulta).

Byz:))))).

Attenzione

Questo tutorial è a solo scopo informativo. Ne io, ne i possessori del sito, siamo responsabili di un eventuale abuso di ciò che è stato trattato, per scopi di pirateria e violazione di copyright.

In qualsiasi caso rubare è sbagliato, e i vari tutorial presenti sul sito dovrebbero aiutare a comprendere l'impegno profuso da parte dei programmatori, per la creazione d'ogni singolo programma.