



# StackGuard

buffer overflow defense

a cura di

Bartalotta Andrea	56/100083
Capone Angelo	56/100034
Capozza Michele	56/100113
Ferri Guglielmo	56/00991
Piccolo Sabatino	56/00983

a cura di



## Introduzione

- ✓ Il **Buffer Overflow** è una delle forme più comuni di vulnerabilità di sistemi.
- ✓ Domina l'area relativa alla penetrazione in reti remote.
- ✓ **StackGuard** - sviluppato nei laboratori dell'università dell'oregon. Si prefigge di rendere immune il sistema da questo tipo di attacco.



## Gestione del Buffer

- **Buffer** - blocco contiguo di memoria che contiene più istanze dello stesso tipo di dato.
- **Overflow** - riempire oltre il limite un buffer.
- Come è organizzata la memoria di un processo?

variabili statiche e dati  
(inizializzate e non)



- fissata
- a sola lettura
- codice del programma



## Cos'è uno Stack

- Tipo di dato astratto che gode della proprietà LIFO.
- Operazioni principali - *push* e *pop*.
- Appare come un blocco di memoria contiguo contenente dei dati.
- **Stack pointer (SP)** - registro che punta alla cima dello stack.
- La **base** è un indirizzo fisso.



Linguaggi di programmazione ⇒ costruito di procedura o funzione.

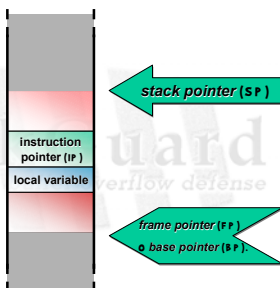


Implementata ad hoc con il supporto di uno stack.



## Cos'è uno Stack ( 2 )

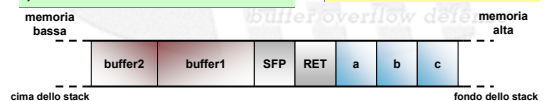
stack frame



## Un esempio: la regione dello Stack

```
void function(int a, int b, int c){
  char buffer1[245];
  char buffer2[512];
}
void main(){
  function(1,2,3);
}
```

- *call* imposta l'istruzione pointer
- *IP* viene chiamato indirizzo di ritorno
- pushl \$3
- pushl \$2
- pushl \$1
- call function



SFP - saved frame pointer, è il vecchio frame pointer salvato quando viene avviata la funzione.  
 RET - l'indirizzo di ritorno.

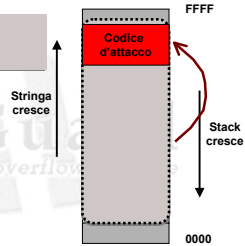


## Cos'è un attacco di *Buffer Overflow*

L'attaccante fornisce una stringa di grosse dimensioni ad un programma che *non effettua controlli sulla taglia dei propri input*.

La stringa sovrascrive l'indirizzo di ritorno e inietta il codice.

La funzione ritorna "saltando" al codice iniettato.



Buffer Overflow

7



## Come attaccare con *Buffer overflow*



Deve *solo* trovare *codice vulnerabile* all'interno di programmi residenti sul sistema.

### ● *Codice vulnerabile*

Qualsiasi operazione effettuata su di un array che non effettua controlli sulla dimensione.

● E esistono *cook-books* da cui estrarre il codice necessario.

Buffer Overflow

8



## *Buffer Overflow: un problema attuale*

➢ Grande risonanza nel 1988 dopo il *Worm* di Internet (Morris).

➢ Non c'è stato nessun segno di miglioramento, in quanto:

- causato da funzioni "pericolose" del C presenti nelle librerie standard (*strcpy()*, *strcat()*, etc).
- *solo* programmazione accurata può limitare questa vulnerabilità.
- nuove patch introducono nuove vulnerabilità.
- un *debugging* accurato potrebbe non eliminare il problema.

Buffer Overflow

9

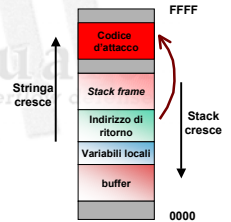


## Vulnerabilità ed attacchi

• Lo scopo è di sovvertire la funzione di un programma privilegiato.

• Per ottenere questo risultato l'attaccante deve:

1. predisporre il codice adatto, da eseguire nello spazio d'indirizzamento del programma.
2. permettere al programma di saltare a quel codice, con parametri esatti, caricati nei registri e nella memoria.



Buffer Overflow

10



## Come utilizzare codice d'attacco

Due modi per realizzare un *buffer overflow*:

### ❖ *Iniettare il codice:*

- L'attaccante fornisce una stringa in input.
- Il programma lo carica in un buffer.
- L'attaccante sta usando i buffer del programma vittima per memorizzare il codice d'attacco.

### ❖ *Il codice è già lì:*

- Il codice adatto è presente nello spazio di indirizzamento.
- L'attaccante ha bisogno di parametrizzare il codice.
- L'attaccante fa in modo che il programma salti ad esso.

Buffer Overflow

11



## Modifica del flusso

Corrompere i puntatori ⇒ *indirizzo di ritorno*

### ❖ *Record d'Attivazione:*

- Corrompere l'indirizzo di ritorno nel record di attivazione.
- L'attaccante causa il salto del programma al codice d'attacco.
- *stack smashing attack* (attacco che fraccassa lo stack).

### ❖ *Puntatori a Funzione:*

- Possono essere allocati ovunque (stack, heap, area dati).
- L'attaccante necessita di trovare un buffer adiacente al puntatore a funzione.
- Mandare queste aree in overflow per cambiare il puntatore.

Buffer Overflow

12



## Tecnica tipica d'attacco

Combinazione di:

- tecnica d'iniezione.**
- corruzione del record d'attivazione.**

Nota:

- **Iniezione e Corruzione non avvengono in un'unica azione.**
- **L'attaccante può iniettare il codice oppure far traboccare un buffer differente per corrompere il puntatore al codice.**
- **Se il codice è già residente ⇒ L'attaccante ha bisogno di parametrizzarlo.**

Buffer Overflow

13



## Difese da Buffer Overflow

Quattro approcci di base:

- ✓ **Il metodo di forza bruta ovvero scrivere codice sicuro.**
- ✓ **Rendere l'area di memoria, destinata a contenere le variabili, non eseguibile.**
- ✓ **Controllare la dimensione degli array ad ogni accesso.**
- ✓ **Verificare l'integrità dei puntatori prima di dereferenziarli.**

Buffer Overflow

14



## Scrivere codice sicuro

- **Irrmediabilmente costoso!**
- **utilizzare tools come grep.**
- **Programmazione sicura dovrebbe seguire le seguenti regole:**
  - **Principio del minor privilegio possibile.**
  - **Scrivere codice semplice.**
  - **Non fidarsi di nessuno.**



**Vulnerabilità al buffer overflow possono essere non facilmente individuabili**

Buffer Overflow

15



## Buffer non eseguibili

**Impedire all'attaccante di eseguire il codice inserito**

⇒ **Rendere il segmento dati dello spazio di indirizzamento non eseguibile.**

### Problemi

- **Linea di progettazione dei sistemi sui vecchi computer**
- **I più recenti sistemi UNIX e MS Windows, dipendono dalla possibilità di inserire codice dinamico nel segmento dati dei programmi, per ottimizzare le prestazioni.**
- **Alcuni sistemi devono sacrificare sostanzialmente compatibilità dei programmi già in uso.**



- patch per Linux e Solaris implementano questo criterio.**
- pochi problemi di compatibilità:**
  - **nessun programma "normale" ha del codice eseguibile nello stack.**

Buffer Overflow

16



## Controllo della dimensione degli array

Non basta inserire codice per realizzare un buffer overflow:

➢ **è necessario modificare il flusso del programma in esecuzione.**

- ❖ **Non è possibile corrompere i dati adiacenti nello stack.**
- ❖ **Elimina completamente le possibilità di realizzare tali attacchi.**
- ❖ **L'approccio diretto è quello di testare tutti i riferimenti agli array.**

Realizzazioni:

- ✓ **Controllo sulla dimensione degli array: (Jones & Kelly).**
- ✓ **Controllo dell'accesso in memoria.**
- ✓ **Linguaggi Type-Safety.**

Buffer Overflow

17



## Verifica dell'integrità dei puntatori

Verifica:

➢ **un puntatore è stato sovrascritto prima di essere utilizzato?**



**Attacchi mirati alla modifica di componenti del programma, che non siano i puntatori, andranno comunque a segno.**

**sviluppato in 3 modi distinti:**

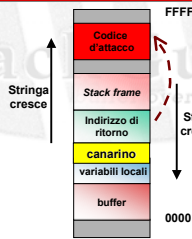
- **Snarskii - versione personalizzata di libc per FreeBSD.**
- **Progetto StackGuard.**
- **PointGuard - in fase di sviluppo.**

Buffer Overflow

18

## StackGuard

Realizzato utilizzando una tecnica di compilazione che assicura l'integrità dell'area di memoria contenente l'indirizzo di ritorno del record d'attivazione della funzione.



Buffer Overflow

19

## StackGuard ( 2 )

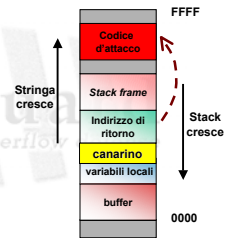
✓ StackGuard è una piccola patch da applicare a gcc.

➢ aggiunge del codice specifico.

✓ Il codice aggiuntivo piazza una **canary word** vicino all'indirizzo di ritorno nello stack.

✓ Il codice verifica che la **canary word** sia intatta.

➢ dopo salta all'indirizzo di ritorno!



Buffer Overflow

20

## StackGuard ( 3 )

**Problema:** un attaccante legge la **canary word** e la incapsula nella stringa di attacco.



Può sovrascrivere l'indirizzo di ritorno senza compromettere **canary word**.

**Soluzione:**

StackGuard utilizza tre metodi per prevenire questo attacco:

- Terminator Canary
- Random Canary
- Xor Random Canary

Buffer Overflow

21

## Terminator Canary

**canary word** costituita da comuni simboli di terminazione utilizzati dalle librerie standard del C:

- 0 - NULL
- CR - carriage return
- LF - Line Feed
- 1 - End of file

**Nota che:**

- Un maintenzionato non potrebbe usare le funzioni standard per leggere questi simboli ed incapsularli in una stringa.
- Le funzioni di copia si fermerebbero alla prima occorrenza di uno di questi caratteri.

Buffer Overflow

22

## Random Canary

**canary word** di 32 bit scelto al run-time.

- Segreto facile da usare difficile da indovinare.
- Non è mai rivelata.
- Cambia ad ogni riavvio del programma.

Buffer Overflow

23

## Xor Random Canary

Mecanismo introdotto dalla versione 1.21 di StackGuard.

➢ **canary word** consiste di un vettore di 128 bit casuali (quattro word scelte al run-time)

➢ XOR (4 word, return address).



**canary word** legata all'indirizzo di ritorno della funzione attiva.

Buffer Overflow

24

## Resistenza alle intrusioni

Programma Vulnerabile	Risultato senza StackGuard	Risultato con StackGuard
PIP 3.3.7n	Root shell	Programma bloccato
Elim 2.4 PL25	Root shell	Programma bloccato
Perl 5.003	Root shell	Programma bloccato con uscita irregolare
SamBA	Root shell	Programma bloccato
SuperP r08e	Root shell	Programma bloccato con uscita irregolare
umont2.Sk/Bc 5.3.12	Root shell	Programma bloccato
wwwcount v2.3	HTTP shell	Programma bloccato
ZGV 2.7	Root shell	Programma bloccato

Buffer Overflow

25

## Risultati sperimentali

- Sperimentalmente StackGuard realizza *protezione effettiva* contro attacchi di *stack smashing*.
- Preserva caratteristiche di compatibilità e utilizzo del sistema protezione.
- Progetto *Immunix*: compilazione di un'intera distribuzione di Linux (Red Hat 5.1) con StackGuard.
- *microbenchmark* hanno evidenziato sostanziale incremento nel costo di ogni singola chiamata a funzione.
- *macrobenchmark* hanno evidenziato un *overhead* totale trascurabile.

Buffer Overflow

26

## Risultati sperimentali (2)

Testo di StackGuard sul web server Apache

StackGuard	# di client	Connessioni (per secondo)	Latenza media (in secondi)	Throughput (in MBit/sec)
No	2	34.44	0.0578	5.63
No	16	43.53	0.3583	6.46
No	30	47.2	0.6030	6.46
SI	2	34.92	0.0570	5.53
SI	16	53.57	0.2949	6.44
SI	30	50.89	0.5612	6.48

Buffer Overflow

27

## Considerazioni finali

- StackGuard opera in modo del tutto trasparente all'utente.
- Sul sito di Immunix si fa riferimento ad una versione 2.0 di StackGuard, tutt'oggi risulta irripetibile.
- Le patch disponibili sono relative a versioni obsolete del compilatore gcc.
- L'entusiasmo iniziale nei confronti di questo progetto è andato scemando.

Buffer Overflow

28

## Laboratorio



Buffer Overflow

29

## Exploit & Shellcode

- ✓ L'exploit sfrutta una vulnerabilità presente in un sistema.
- ✓ Il nostro exploit genera la stringa che sovrascrive l'indirizzo di ritorno ed esegue il codice di attacco.
- ✓ Una shellcode è un array di caratteri che contiene il codice eseguibile da iniettare.

Esempio

```
char shellcode[] =
"\xeb\x2a\x5e\x89\x76\x08\xc6\x46\x07\x00\xc7\x46\x0c\x00\x00\x00"
"\x00\xb8\x0b\x00\x00\x00\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80"
"\xb8\x01\x00\x00\x00\xb8\x00\x00\x00\xcd\x80\xe8\xcd\xff\xff"
"\xf3\xf2\x62\x69\x6e\xf3\xf7\x68\x00\x89\xec\x5d\xc3";
```

Buffer Overflow

30

