



Quali sono oggi le probabilità di accedere da remoto ad un sistema Linux sfruttando un comune stack overflow? Negli ultimi anni le misure di sicurezza volte a contrastare questo genere di minaccia sono aumentate esponenzialmente nel tentativo di frenare il fenomeno della propagazione dei worm e ridurre le possibilità di un aggressore esterno di condurre con successo attacchi intrusivi. Reperire infatti exploit “**one-shot**” (compila, esegui ed ottieni una shell al primo colpo) soprattutto per il pinguino è diventato abbastanza arduo. Seppure esistano tecniche più o meno efficienti per aumentare le probabilità di successo di un attacco contro le tecnologie **anti-overflow** oggi esistenti, le possibilità di portarlo a termine in modo corretto si abbassano notevolmente quando più fra queste tecnologie vengono implementate a protezione di un sistema. Se è vero che “*l'unione fa la forza*” è proprio questo il concetto a cui molti vendor Linux si sono ispirati. Sono diverse infatti le distribuzioni (a parte ovviamente qualche rara eccezione) che applicano al kernel o alla librerie di sistema un mix di patch differenti per la sicurezza, allontanandolo così considerevolmente dalla sua originaria versione **vanilla**. In questo contesto sono quattro le tecnologie comunemente implementate che è importante conoscere. Diamo loro brevemente uno sguardo per comprendere quali barriere antepongono al corretto svolgimento di un attacco intrusivo.

### No-eXecute

E' una funzionalità supportata dai moderni processori AMD ed Intel che si riferisce in realtà all'ultimo bit (il bit **63**) della tabella di paging, il cui scopo è quello di marcare le aree di memoria a seconda degli effettivi utilizzi. Ad esempio marcando lo **stack** e l'**heap** come non eseguibili, viene preclusa la possibilità di inserire in queste aree di memoria uno **shellcode** che possa essere richiamato alterando l'indirizzo di ritorno di una funzione o un puntatore generico. In questo modo la parte in cui sono contenuti i **dati** può essere separata dai segmenti che contengono invece **istruzioni eseguibili** dal processore:

```
# cat /proc/2306/maps
[...]
08048000-08049000 r-xp 00000000 fd:00
621089      /usr/bin/server

08049000-0804a000 rw-p 00000000 fd:00
621089      /usr/bin/server

[...]
bfd26000-bfd3b000 rw-p bfd26000 00:00
0          [stack]
```

In questo esempio, ottenuto attraverso il filesystem `/proc`[1], è possibile osservare i permessi assegnati alle varie aree di memoria dell'applicazione “`/usr/bin/server`”. Come si può notare lo stack risulta essere solamente leggibile e scrivibile. Il contenuto al suo interno verrà quindi interpretato unicamente come **dati**. Il contenuto presente invece fra l'indirizzo `0x08048000` e `0x08049000` verrà interpretato dal processore come **istruzioni** eseguibili (notare i permessi `r-xp`). Nel nostro caso questa zona rappresenta il codice compilato del file binario e mappato in memoria. Poiché la funzionalità **No-eXecute** non è supportata da tutti i processori (in particolare è assente in modo nativo su piattaforma x86), alcuni sistemi operativi la emulano a livello software. Per Linux esistono diverse implementazioni. Le più famose sono **PaX**[2] ed **Exec-Shield**, quest'ultima sviluppata da Red Hat.

---

[1] `cat /proc/pid/maps`

Per osservare come da sistema operativo Linux vengono marcate le aree di memoria di un'applicazione già in esecuzione è necessario determinare il pid del programma attraverso il comando `ps` o simile e visualizzare da filesystem `/proc` il file `maps`

[2] <http://www.grsecurity.net/>





## Address Space Layout Randomization (ASLR)

E' una tecnica che permette di randomizzare lo spazio di indirizzamento virtuale di un programma. Ad ogni esecuzione di un'applicazione l'indirizzo base dello **stack**, dell'**heap**, delle **librerie linkate** e/o della stessa regione di memoria in cui il file eseguibile viene mappato, variano randomicamente. In questo modo diviene difficile predire con esattezza il punto in cui uno shellcode verrà collocato in memoria o l'indirizzo in cui si troveranno puntatori e/o funzioni di interesse, riducendo notevolmente l'efficacia sia delle classiche tecniche di buffer overflow che di quelle più avanzate (return-to-libc, VDSO return-to-stack, etc...). Le aree di memoria interessate dalla randomizzazione variano a seconda della configurazione del sistema, delle patch del kernel applicate dalla distribuzione in uso e/o dalle opzioni di compilazione utilizzate. Si osservi il seguente esempio:

```
# gdb ./server -q
(gdb) break *main
Breakpoint 1 at 0x8048604
(gdb) run
Breakpoint 1, 0x08048604 in main ()
(gdb) p $esp
$1 = (void *) 0xbfbe362c
(gdb) p system
$2 = {<text variable, no debug
info>}0x1c8550 <system>
(gdb) quit
```

Appena dopo l'ingresso nella funzione `main()` dell'applicazione `server`, lo **Stack Pointer** punta all'indirizzo di memoria `0xbfbe362c` mentre la funzione `system()` può essere raggiunta attraverso l'indirizzo `0x1c8550`. Ma eseguendo più volte l'applicazione e ripetendo la sessione di debugging con `gdb`<sup>[3]</sup> questi indirizzi variano di continuo:

```
(gdb) p $esp
$1 = (void *) 0xbf9efec
(gdb) p system
$2 = {<text variable, no debug
```

```
info>}0x4f6550 <system>
[...]

[...]
(gdb) p $esp
$1 = (void *) 0xbffdd21c
(gdb) p system
$2 = {<text variable, no debug
info>}0x788550 <system>
```

L'**Address Space Layout Randomization** è attivo di default sul kernel Linux a partire dalla versione 2.6.20 ma diverse distribuzioni la implementato da tempo. Ad esempio Red Hat ha sviluppato la propria tecnologia che fa parte di Exec-Shield. Anche **PaX** offre questo tipo di supporto. Comunemente ASLR può essere attivato/disattivato attraverso `filesystem /proc`<sup>[4]</sup>

## Stack Canary

E' la tecnologia **anti stack overflow** su cui gli esperti ripongono maggiori speranze. Consiste nel generare, ad ogni esecuzione di un'applicazione, un valore (randomico o statico a seconda delle implementazioni e delle circostanze) definito **canary**. Il canary (Figura 1), in fase di costruzione dello stack frame di ogni funzione che utilizzi buffer più grandi di una certa dimensione (solitamente dai 4/8 byte in su), viene collocato subito dopo gli eventuali puntatori ed i buffer allocati ma prima del **Frame Pointer** (puntato dal registro **EBP**) e dell'indirizzo di ritorno della funzione (puntato dal registro **EIP**). Il canary viene poi controllato subito prima dell'invocazione dell'indirizzo di ritorno. Se il suo valore differisce dall'originario allora la causa è da imputare ad una condizione di overflow (Figura 2) che lo ha impropriamente sovrascritto e pertanto l'applicazione viene terminata con il segnale `SIGABRT`.

[3] <http://www.gnu.org/software/gdb/gdb.html>

[4] `echo 0 > /proc/sys/kernel/randomize_va_space` disattiva ASLR.  
`echo 1 > /proc/sys/kernel/randomize_va_space` attiva ASLR.





Figura 1: collocazione del canary nello stack frame di una funzione

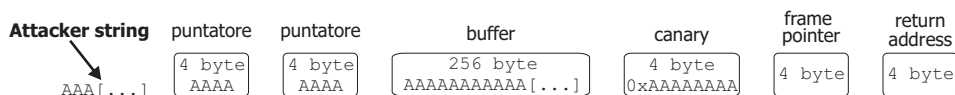


Figura 2: Un overflow sovrascrive puntatori, buffer e canary in memoria. `__stack_chk_fail` al ritorno della funzione di copia si accorgerà di questa situazione evitando che il flusso di esecuzione del programma riprenda dal return address

Si comporta in questo modo **SSP** (meglio noto come **ProPolice**) che attraverso una patch per GCC è implementato di default a partire dalla versione 4.1 del compilatore GNU. Altre implementazioni, come nel caso delle prime versioni di **StackGuard**, possono comportarsi in modo diverso (ad esempio proteggendo solo l'indirizzo di ritorno di una funzione ma non il suo Frame Pointer).

In alcune distribuzioni Linux come **Fedora**, per attivare la protezione offerta da SSP, è necessario specificare esplicitamente l'opzione `-fstack-protector` quando si compila dai sorgenti. In altre invece (ad esempio nelle ultime edizioni di **Ubuntu**) questa è implicitamente attiva e per disattivarla è necessario specificare l'opzione `-fno-stack-protector`. Per conoscere se un'applicazione è stata compilata con la protezione SSP è possibile utilizzare lo strumento `objdump`<sup>[5]</sup> e verificare la presenza della funzione `__stack_chk_fail`:

```
# objdump -d ./s | grep stack
0804859c <__stack_chk_fail@plt>:
 8048977:      e8 20 fc ff ff
 call   804859c <__stack_chk_fail@plt>
```

La funzione `__stack_chk_fail` si occupa di verificare la congruità del canary ed è uno dei primi blocchi di codice da analizzare se si vuole comprendere il funzionamento di SSP sotto Linux. Le tecnologie **Stack Canary** non sono in grado di proteggere da condizioni di overflow che si verificano in altre regioni di memoria al di fuori dello Stack (ad esempio nell'**Heap**).

### FORTIFY SOURCE

**FORTIFY\_SOURCE** è una nuova feature aggiunta in alcune distribuzioni sotto forma di patch al tree di sorgenti del GCC<sup>[6]</sup> che può essere utilizzata per rilevare e prevenire alcuni tipi di buffer overflow (stack, heap e format string) prima che si verifichino.

```
void funzione(char *string)
{
    char buf[20];
    strcpy(buf, string);
}
```

Nell'esempio di codice sopra evidenziato, il compilatore conosce in anticipo la dimensione del buffer `buf` nello stack e può quindi prevenire che più di 20 byte di dati vengano copiati al suo interno. Questo tipo di controllo viene effettuato da apposite funzioni di check a seconda della funzione di copia utilizzata nel programma. Ad esempio nel caso appena mostrato un'analisi del binario rivelerà una o più chiamate a `__strcpy_chk` subito dopo l'invocazione di ciascuna `strcpy()`:

```
# objdump -d server | grep strcpy
[...]
804865a: e8 a1 fe ff ff  call 8048500
                <__strcpy_chk@plt>
80487a8: e8 53 fd ff ff  call 8048500
                <__strcpy_chk@plt>
```

[5] <http://directory.fsf.org/binutils.html> - Lo strumento fa parte delle GNU binutils. E' incluso di default in tutte le principali distribuzioni.

[6] <http://directory.fsf.org/gcc.html> - Compilatore GNU per i linguaggi C/C++/Java/Ada/Fortran. Costituisce la base degli strumenti di sviluppo di ogni distribuzione Linux.





FORTIFY\_SOURCE aggiunge funzioni di check per la maggior parte delle funzioni preposte allo spostamento di dati da un'area di memoria all'altra che possono causare un buffer overflow come `memcpy`, `memmove`, `memset`, `strcat`, `strncat`, `strncpy`, etc..., incluse quelle che permettono di indicare il formato della stringa (`printf`, `fprintf`, `vfprintf`, `snprintf`, `sprintf`, `vsnprintf` e `vsprintf`)

### Riepilugum meglium est

In **tabella 1** è riportato il riepilogo schematizzato delle funzionalità di sicurezza supportate di default dalle ultime distribuzioni Linux esistenti nel momento in cui si scrive. **Debian** e **Slackware** che tendono ad avvicinarsi il più possibile al kernel vanilla (ovvero privo di patch aggiuntive), risultano essere i sistemi meno complessi da violare, mentre tra i più problematici rientra senza dubbio **Fedora**.

Alcune fra le tecnologie anti overflow introdotte nei paragrafi precedenti e le metodologie note per bypassarle verranno descritte in dettaglio già a partire da questo numero della rivista.

### E' la fine dell'hacking?

La maggior parte delle tecnologie anti overflow sviluppate in questi ultimi anni sono state integrate di default nelle moderne distribuzioni Linux solo a partire dal kernel 2.6. Ciò significa che tutti i sistemi con versioni inferiori ne sono potenzialmente sprovviste<sup>[7]</sup>. Se da un lato è possibile obiettare che tutte le principali distribuzioni desktop e server sono oramai passate al kernel 2.6, dall'altro è possibile affermare che in ambienti di produzione questo switch non è ancora avvenuto in modo completo e definitivo. Si osservi a tal proposito la **tabella 2** sviluppata a partire da alcuni dati raccolti effettuando fingerprint<sup>[8]</sup> remoto su un campione di 1000 sistemi connessi ad Internet. La tabella consta di tre colonne che indicano la percentuale di kernel di tipo **2.2**, **2.4** e **2.6** rilevata durante il probing e di altrettante righe che

indicano i comparti territoriali a cui tali percentuali si riferiscono (Centro Europa, Cina e Stati Uniti). Lo schema evidenzia chiaramente un'alta percentuale di server Linux con kernel 2.4 ancora attivi. Tale circostanza è sorprendentemente più marcata negli Stati Uniti che in Cina, dove comunque la differenza a favore dei kernel 2.6 è appena dello 1,2% e dove i sistemi con kernel 2.2 rappresentano addirittura il 23,38% dell'intero campione sondato. Pur trattandosi di dati statistici ricavati a partire da un range limitato di host rispetto alla quantità di indirizzi IP disponibili ed allocati in internet, rimane però immutabile il fatto che in rete un'alta percentuale di server è ancora sprovvista di ogni sorta di protezione anti overflow, il che rende questi sistemi dei target particolarmente appetibili e senza dubbio più accessibili rispetto a quelli in cui tali tecnologie sono invece già presenti. Verrebbe dunque da asserire che l'avvento dei kernel 2.6 sta accompagnando le tecniche di hacking classico (quelle fatte cioè di buffer sovrascritti oltre le massime capacità di contenimento consentite, di puntatori deviati e di shellcode collocati in aree di memoria propizie) verso il tramonto. In realtà esistono ancora buoni margini di successo per hacker e worm-writer di scrivere exploit funzionanti, seppure le percentuali di fare breccia ai primi tentativi si siano sensibilmente ridotte con le nuove tecnologie anti overflow.

I due articoli pratici che seguono nelle pagine successive hanno proprio lo scopo di far comprendere la differenza che intercorre tra violare un vecchio sistema con kernel 2.4 e violarne uno più moderno con kernel 2.6 e con le ultime funzionalità di sicurezza implementate.

---

[7] PaX può essere applicato manualmente anche su svariate versioni del kernel branch 2.4 e 2.2.

[8] si tratta di una tecnica che si avvale di richieste TCP, UDP e/o ICMP, definite **probe**, per identificare con approssimativa certezza il sistema operativo di un server e/o client connesso in rete.





## COME SONO STATI GENERATI I DATI DELLA TABELLA 2 ?

Le statistiche della **Tabella 2** sono state generate utilizzando due tecniche di fingerprint distinte basate principalmente su protocolli ICMP e TCP, rispettivamente implementate dagli strumenti **xprobe2**<sup>[9]</sup> ed **nmap**<sup>[10]</sup>.

### Perché due diversi strumenti?

La scelta di utilizzare due diversi strumenti è stata dettata dalla necessità di produrre risultati il più possibile consistenti e congruenti. Poiché il traffico ICMP viene solitamente filtrato dai firewall, si è scelto di affiancare alla scansione di **xprobe2** quella di **nmap**, così da procedere con una più accurata identificazione del kernel dei sistemi testati. La tecnica del TCP Fingerprint implementata da Nmap è però molto dispendiosa in quanto necessità di almeno una porta aperta ed una chiusa per poter produrre risultati completi. Se una di queste condizioni non viene soddisfatta, il tool procede alla scansione automatica di tutte le porte TCP dell'host remoto. Laddove il traffico ICMP non fosse filtrato ed il risultato finale potesse essere reputato sufficientemente attendibile, si è preferito quindi lavorare esclusivamente con **xprobe2** che fa uso di un numero di richieste più circoscritto e che pertanto consuma meno banda di rete, generando di conseguenza meno traffico e non urtando la sensibilità dei sistemi IDS. Solo per gli indirizzi IP per i quali non è stato possibile produrre alcun risultato congruo si è proceduto ad un'ulteriore verifica con **nmap**.

### Come è stato selezionato il campione di IP per il test?

Il range di indirizzi IP da testare è stato determinato consultando l'attuale lista dei blocchi di assegnazione IPv4 dal sito dello IANA<sup>[11]</sup>. Per semplicità i blocchi contraddistinti dalla dicitura "Various Registries" sono stati scartati. Il range limitato di IP costituente il campione sondato è stato invece selezionato a partire dai blocchi **esclusivamente** assegnati ad uno dei tre registri facenti capo all'area Europea, Asiatica e delle Americhe (rispettivamente RIPE, APNIC ed ARIN). Per ciascuno di questi tre comparti territoriali è stato effettuato uno **Sweep Scan** con Nmap per identificare gli host attivi nel range:

```
# nmap -sP -n -oG IP_active_asia.txt  
211.157.0-10.*
```

Dove:

-sP attiva la modalità **Sweep Scan** (ovvero procede con l'invio di un **icmp echo request** e di un pacchetto **TCPACK** per determinare se la destinazione è online).

-n disattiva la risoluzione DNS degli host.

-oG stampa l'output nel file `IP_active_asia.txt` in un formato facilmente manipolabile con il comando `grep` (ovvero tutte le informazioni ricavate per ciascun indirizzo IP vengono memorizzate sulla stessa riga).

`211.157.0-10.*` rappresenta uno dei range IP scelto per il sondaggio (nell'esempio da 211.157.0.0 a 211.157.10.255).

Dall'output generato con **nmap** è stata poi successivamente ricavata la lista degli indirizzi IP attivi, sgrondandolo delle informazioni inutili:

```
# cut -d " " -f 2 IP_active_asia.txt |  
grep -v Nmap > asia_list1
```

La parte del comando prima del pipe (`|`) serve a stampare per ogni riga di testo presente nel file `IP_active_asia.txt` il contenuto della seconda colonna separato dal delimitatore spazio. L'altra parte del comando rimuove invece dall'output scritto nel file `asia_list1` la stringa "Nmap". Ciò è sufficiente per generare una lista di IP attivi utilizzabile come input delle fasi successive.

### Una volta determinati strumenti da utilizzare e campione da sondare, come è stato svolto il test?

Per le scansioni con **xprobe2** è stato utilizzato il seguente script bash:

```
#!/bin/sh  
  
IPS=`cat $1`  
  
for i in $IPS  
do  
xprobe2 $i -M 6 -M 7 -M 8 -M 9 -M 10 >>  
$2_xprobe2_fingerprint.txt
```





done

che eseguito nel modo di seguito indicato:

```
# chmod +x script
# ./script IP_active_asia.txt asia
```

genera un unico file denominato `asia_xprobe2_fingerprint.txt` in cui viene scritto tutto l'output di `xprobe2`.

Per le scansioni con `nmap` l'intera attività è invece stata svolta in modo più semplice utilizzando direttamente la lista degli indirizzi IP ricavata nella fase precedente:

```
# nmap -iL IP_active_asia.txt -n -P0 -O
-oN asia_nmap_fingerprint.txt
```

Dove:

`-iL` Indica ad `nmap` di prelevare gli indirizzi IP da sondare direttamente dal file lista `IP_active_asia.txt`.

`-P0` Salta i test per verificare l'effettiva raggiungibilità

degli host (la lista è infatti composta da IP già identificati come attivi).

-O Attiva la modalità TCP Fingerprint

A questo punto terminata la raccolta dei dati, gli stessi sono stati esaminati e sviluppati fino a giungere alle percentuali allegate in **Tabella 2**. Lasciamo comunque al lettore il compito di sperimentare per conto proprio questa fase. Inviatemi le vostre statistiche all'indirizzo [redazione@segfault.it](mailto:redazione@segfault.it) in formato html o pdf, possibilmente corredate da grafici e/o tabelle, assieme agli strumenti o agli script che avete creato per generarle. I migliori lavori prodotti verranno condivisi sul nostro sito Internet o pubblicati direttamente sulla nostra rivista!

[9] <http://www.sys-security.com>

[10] <http://www.insecure.org>

[11] <http://www.iana.org/assignments/ipv4-address-space> - Da Wikipedia: L'Internet Assigned Numbers Authority è un organismo che ha responsabilità nell'assegnazione degli indirizzi IP.



Per migliorare i risultati delle tue statistiche accedi ai contenuti messi a disposizione dalla redazione su <http://www.segfault.it/SS/001/stats/pack.tar.gz>

	N X Patch (emulazione software)	A S L R	S S P	F O R T I F Y S O U R C E
Fedora Core 5/6/7	✓ (Exec-Shield)	✓ (Exec-Shield)	✓ [*]	✓ [**]
Ubuntu 6.10			✓ [*][***]	
Ubuntu 7.04		✓	✓ [*][***]	
OpenSUSE 10.2		✓	✓ [*]	✓ [**]
Slackware 12		✓		
Debian 4.0r1		✓		
Mandriva Free 2007.1		✓	✓ [*]	✓ [**]

**Tabella 1:** Riepilogo schematizzato delle funzionalità di sicurezza implementate di default su piattaforma x86 dalle principali distribuzioni Linux esistenti.

[\*] Tutti i package di sistema più importanti forniti con la distribuzione sono compilati di default con `-fstack-protector`

[\*\*] Tutti i package di sistema più importanti forniti con la distribuzioni sono compilati di default con `-D_FORTIFY_SOURCE=1` e/o 2

[\*\*\*] Durante la compilazione di qualsiasi sorgente l'opzione `-fstack-protector` è implicitamente attiva.

	kernel 2.2	kernel 2.4	kernel 2.6
Centro Europa (RIPE)	1,3%	37,81%	60,89%
Cina (Asia - APNIC)	23,38%	37,76%	38,96%
Stati Uniti (America - ARIN)	2,22%	57,03%	40,75%

**Tabella 2:** Diffusione degli ultimi tre branch stabili del kernel Linux su un campione di 1000 sistemi sondati

