CORSO DI GESTIONE DELLE RETI DI TELECOMUNICAZIONI

LENST - LABORATORIO DI ELABORAZIONE NUMERICA DEI SEGNALI E TELEMATICA

Principi di sicurezza nelle reti di telecomunicazioni

Leonardo Maccari
maccari@lenst.det.unifi.it

Sicurezza: definizione

La sicurezza viene normalmente immaginata come un *processo*, intendendo un insieme dinamico di attenzioni che rendono la vostra rete sicura. La sicurezza deve essere garantita in tutti gli gli aspetti della vostra rete, da quello fisico a quello di gestione degli utenti.

Leonardo Maccari,
maccari@lenst.det.unifi.

Disponibilità del servizio

Disponibilità

- Il servizio deve essere sempre raggiungibile
- La disponibilità viene violata se siete vittima di un attacco DoS: Denial of Service
- La disponibilità del servizio è la cosa più difficile da garantire:
  - Esistono sempre limiti fisici delle risorse
  - Realizzare un attacco DoS deve costare il più possibile
- La disponibilità si ottiene con una accurata progettazione della rete

Leonardo Maccari,
maccari@lenst.det.unifi.

Disponibilità del servizio

Disponibilità

- Il servizio deve essere sempre raggiungibile
- La disponibilità viene violata se siete vittima di un attacco DoS: Denial of Service
- La disponibilità del servizio è la cosa più difficile da garantire:
  - Esistono sempre limiti fisici delle risorse
  - Realizzare un attacco DoS deve costare il più possibile
- La disponibilità si ottiene con una accurata progettazione della rete

Leonardo Maccari,
maccari@lenst.det.unifi.

Disponibilità del servizio

Disponibilità

- Il servizio deve essere sempre raggiungibile
- La disponibilità viene violata se siete vittima di un attacco DoS: Denial of Service
- La disponibilità del servizio è la cosa più difficile da garantire:
  - Esistono sempre limiti fisici delle risorse
  - Realizzare un attacco DoS deve costare il più possibile
- La disponibilità si ottiene con una accurata progettazione della rete

Leonardo Maccari,
maccari@lenst.det.unifi.

Disponibilità del servizio

Disponibilità

- Il servizio deve essere sempre raggiungibile
- La disponibilità viene violata se siete vittima di un attacco DoS: Denial of Service
- La disponibilità del servizio è la cosa più difficile da garantire:
  - Esistono sempre limiti fisici delle risorse
  - Realizzare un attacco DoS deve costare il più possibile
- La disponibilità si ottiene con una accurata progettazione della rete

Leonardo Maccari,
maccari@lenst.det.unifi.

Disponibilità del servizio

Disponibilità

- Il servizio deve essere sempre raggiungibile
- La disponibilità viene violata se siete vittima di un attacco DoS: Denial of Service
- La disponibilità del servizio è la cosa più difficile da garantire:
  - Esistono sempre limiti fisici delle risorse
  - Realizzare un attacco DoS deve costare il più possibile
- La disponibilità si ottiene con una accurata progettazione della rete

Leonardo Maccari,
maccari@lenst.det.unifi.

Disponibilità del servizio

Disponibilità

- Il servizio deve essere sempre raggiungibile
- La disponibilità viene violata se siete vittima di un attacco DoS: Denial of Service
- La disponibilità del servizio è la cosa più difficile da garantire:
  - Esistono sempre limiti fisici delle risorse
  - Realizzare un attacco DoS deve costare il più possibile
- La disponibilità si ottiene con una accurata progettazione della rete

Leonardo Maccari,
maccari@lenst.det.unifi.

Segretezza dei dati scambiati

Segretezza

- I dati scambiati devono rimanere riservati tra le parti che partecipano allo scambio
- Le reti ethernet permettono, generalmente, di fare *sniffing* dei pacchetti
- Per ottenere segretezza si devono utilizzare algoritmi di crittografia (simmetrici, asimmetrici, distribuiti. . .)

Leonardo Maccari,
maccari@lenst.det.unifi.

Segretezza dei dati scambiati

Segretezza

- I dati scambiati devono rimanere riservati tra le parti che partecipano allo scambio
- Le reti ethernet permettono, generalmente, di fare *sniffing* dei pacchetti
- Per ottenere segretezza si devono utilizzare algoritmi di crittografia (simmetrici, asimmetrici, distribuiti. . .)

Leonardo Maccari,
maccari@lenst.det.unifi.

Segretezza dei dati scambiati

Segretezza

- I dati scambiati devono rimanere riservati tra le parti che partecipano allo scambio
- Le reti ethernet permettono, generalmente, di fare *sniffing* dei pacchetti
- Per ottenere segretezza si devono utilizzare algoritmi di crittografia (simmetrici, asimmetrici, distribuiti. . .)

Integrità dei dati scambiati

Integrità

- I dati devono raggiungere la destinazione senza essere stati modificati
- Si possono modificare dati cifrati senza decifrarli (attacchi di *bit flipping*)
- Per ottenere l'integrità dei dati si devono utilizzare funzioni di *hashing*

Leonardo Maccari,
maccari@lenst.det.unifi.

Integrità dei dati scambiati

Integrità

- I dati devono raggiungere la destinazione senza essere stati modificati
- Si possono modificare dati cifrati senza decifrarli (attacchi di *bit flipping*)
- Per ottenere l'integrità dei dati si devono utilizzare funzioni di *hashing*

Integrità dei dati scambiati

Integrità

- I dati devono raggiungere la destinazione senza essere stati modificati
- Si possono modificare dati cifrati senza decifrarli (attacchi di *bit flipping*)
- Per ottenere l'integrità dei dati si devono utilizzare funzioni di *hashing*

Leonardo Maccari,
maccari@lenst.det.unifi.

Autenticazione del mittente dei dati scambiati

Autenticazione

- Chi riceve un'informazione deve essere sicuro che il mittente è effettivamente quello dichiarato
- I protocolli IP e ethernet permettono di effettuare lo *spoofing* degli indirizzi
- Per ottenere l'autenticazione dei dati si devono utilizzare funzioni di firma digitale

Leonardo Maccari,
maccari@lenst.det.unifi.

Autenticazione del mittente dei dati scambiati

Autenticazione

- Chi riceve un'informazione deve essere sicuro che il mittente è effettivamente quello dichiarato
- I protocolli IP e ethernet permettono di effettuare lo *spoofing* degli indirizzi
- Per ottenere l'autenticazione dei dati si devono utilizzare funzioni di firma digitale

Leonardo Maccari,
maccari@lenst.det.unifi.

Autenticazione del mittente dei dati scambiati

Autenticazione

- Chi riceve un'informazione deve essere sicuro che il mittente è effettivamente quello dichiarato
- I protocolli IP e ethernet permettono di effettuare lo *spoofing* degli indirizzi
- Per ottenere l'autenticazione dei dati si devono utilizzare funzioni di firma digitale

Leonardo Maccari,
maccari@lenst.det.unifi.

Non ripudiabilità dei dati scambiati

Non ripudiabilità

- Chi invia un messaggio non può in seguito negare di averlo mandato
- Importante soprattutto a livello di applicazione, nello scambio di documenti
- Se la firma digitale utilizzata è rilasciata da un'ente autorizzato è del tutto equivalente ad una firma su carta

Leonardo Maccari,
maccari@lenst.det.unifi.

Non ripudiabilità dei dati scambiati

Non ripudiabilità

- Chi invia un messaggio non può in seguito negare di averlo mandato
- Importante soprattutto a livello di applicazione, nello scambio di documenti
- Se la firma digitale utilizzata è rilasciata da un'ente autorizzato è del tutto equivalente ad una firma su carta

Leonardo Maccari,
maccari@lenst.det.unifi.

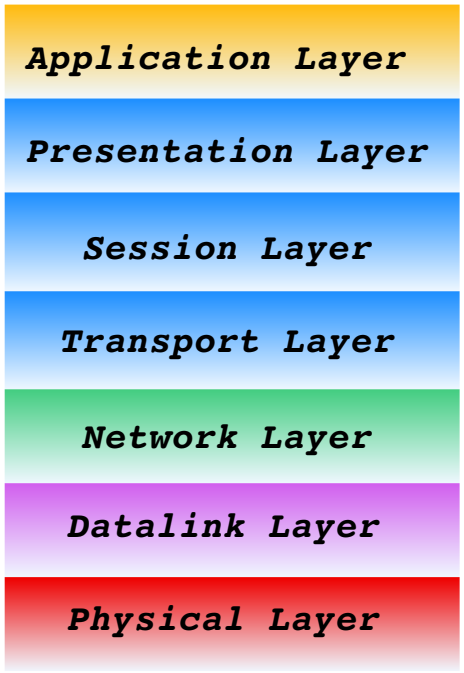
Non ripudiabilità dei dati scambiati

Non ripudiabilità

- Chi invia un messaggio non può in seguito negare di averlo mandato
- Importante soprattutto a livello di applicazione, nello scambio di documenti
- Se la firma digitale utilizzata è rilasciata da un'ente autorizzato è del tutto equivalente ad una firma su carta

Leonardo Maccari,
maccari@lenst.det.unifi.

Stratificazione OSI



Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi al mezzo fisico

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

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi al mezzo fisico

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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 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

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi al livello di collegamento

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

Attacchi al livello di collegamento

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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

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

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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 address space
Protocol address space
[...]
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.

Frame ARP

- campi del protocollo ARP:

Hardware address space
Protocol address space
[...]
Sender hardware address
Sender protocol address
Receiver hardware address
Receiver protocol address

- algoritmo utilizzato:

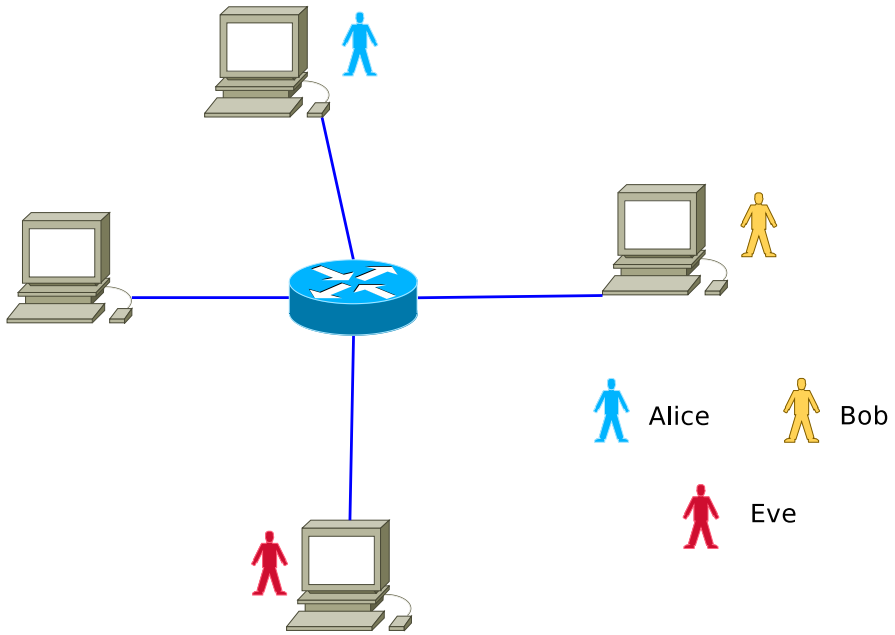
```
?Do I have that hardware type ?
Yes: (almost definitely)
  ?Do I speak that protocol ?
  Yes:
    If the pair <protocol type, sender protocol address> is
        already in my translation table, update the sender
        hardware address field of the entry with the new
        information in the 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.



Leonardo Maccari,
maccari@lenst.det.unifi.

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

Leonardo Maccari,
maccari@lenst.det.unifi.

ARP-Spoofing, segue

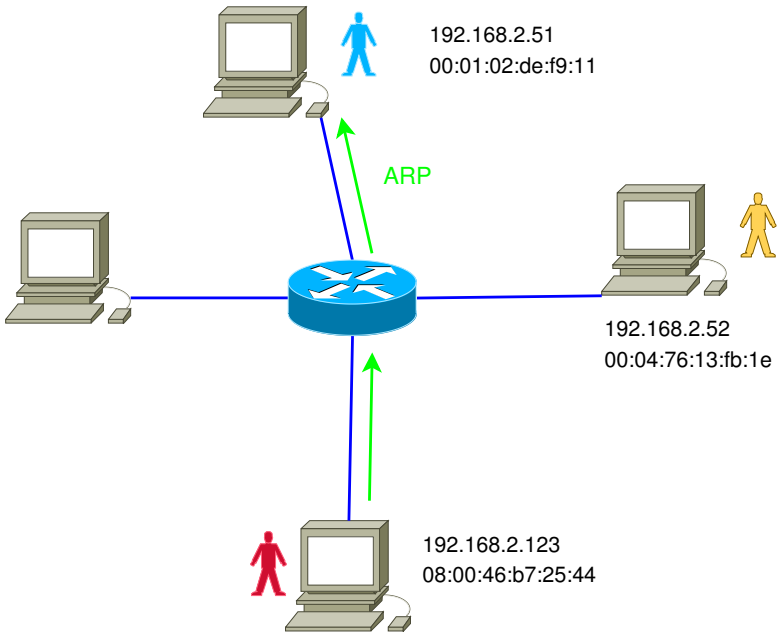
Attacco di ARP-spoofing

- 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-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

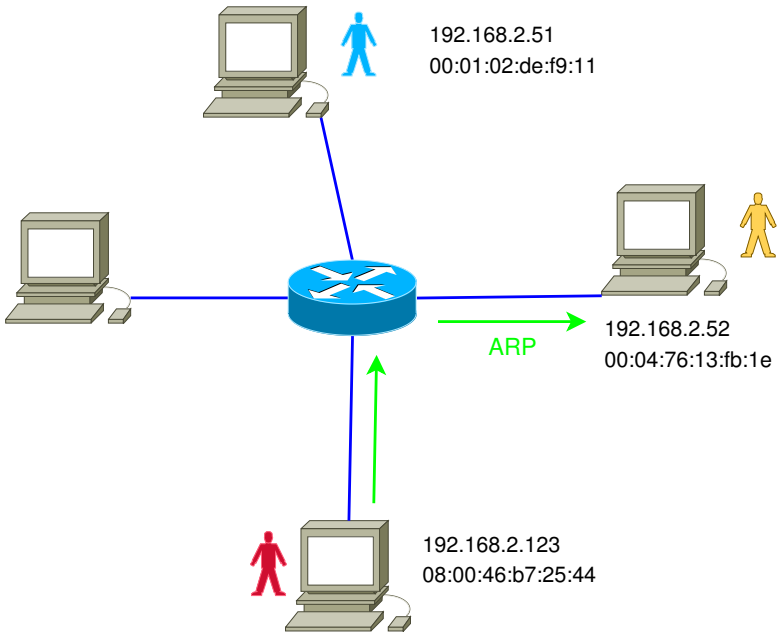


ARP:
192.168.2.52 is at
08:00:46:b7:25:44

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

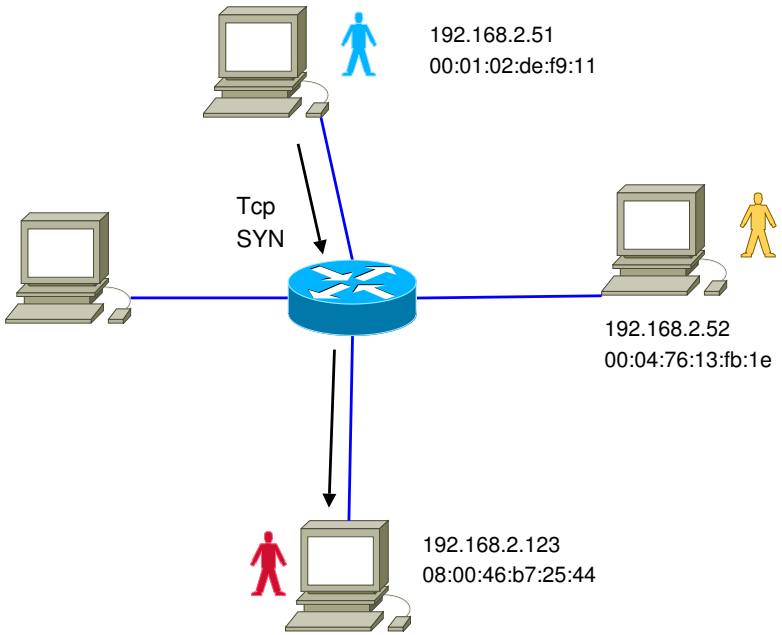


ARP:
192.168.2.51 is at
08:00:46:b7:25:44

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



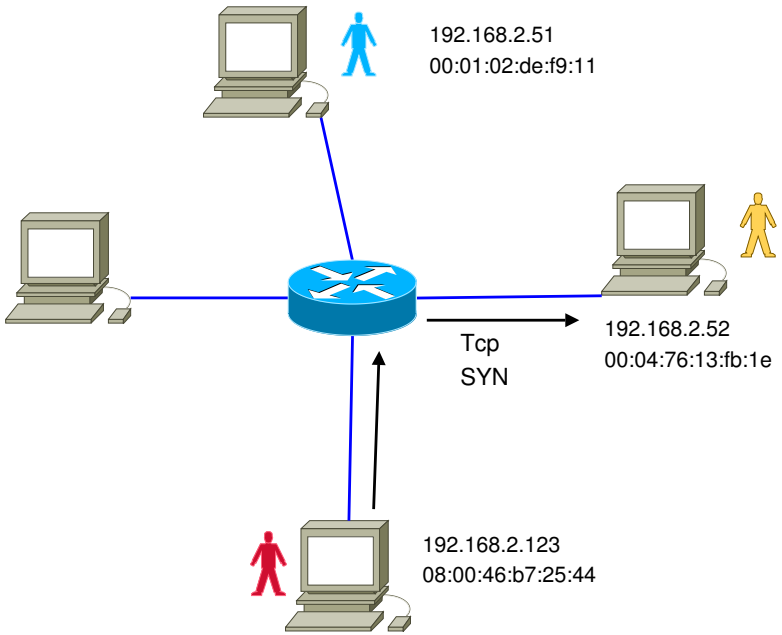
tcp syn:
dst IP 192.168.2.52,
dst MAC 08:00:46:b7:25:44

Leonardo Maccari,
maccari@lenst.det.unifi.

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

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi al livello di rete

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

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi al livello di rete

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

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi al livello di rete

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

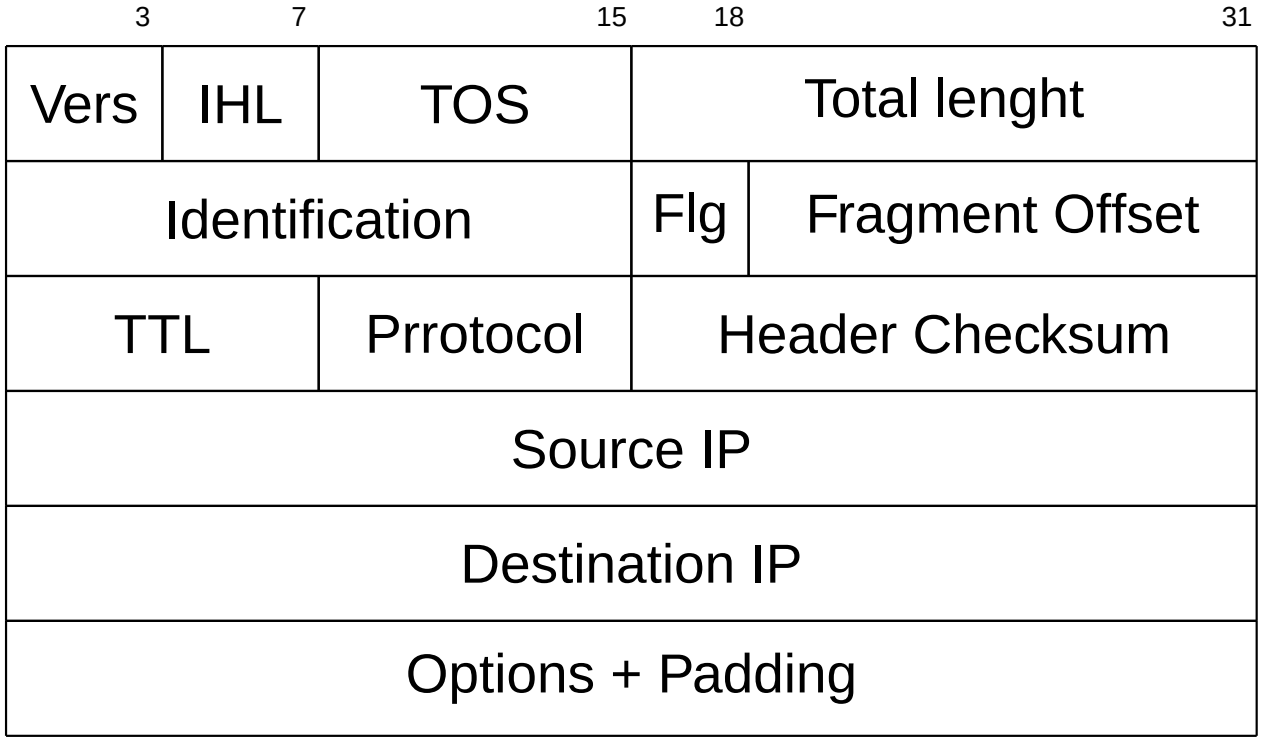
Leonardo Maccari,
maccari@lenst.det.unifi.

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

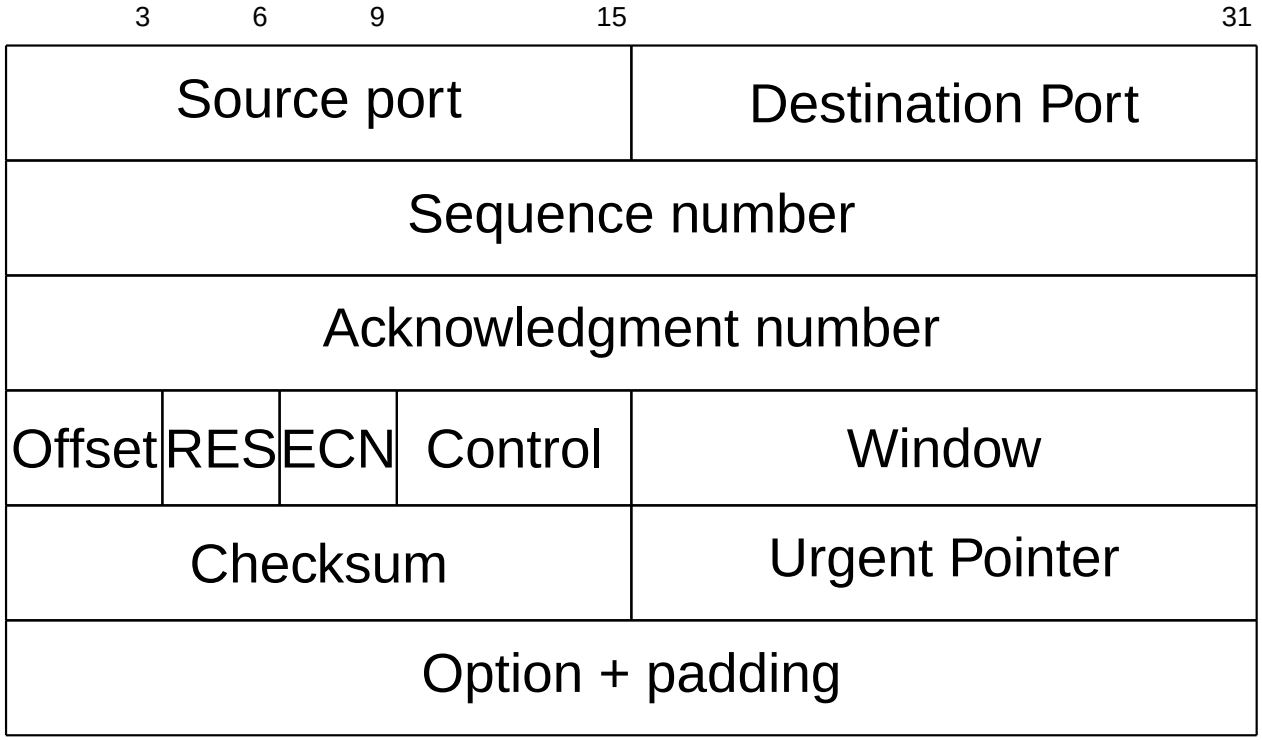
- Il protocollo IP permette di spezzare un pacchetto in piu' frammenti, nel caso questo debba attraversare delle sottoreti con MTU minore della lunghezza del pacchetto stesso.
- **IP Packet Header:**



Fragmentation attacks, segue

Overlapping fragments attack

- **TCP Packet Header:**

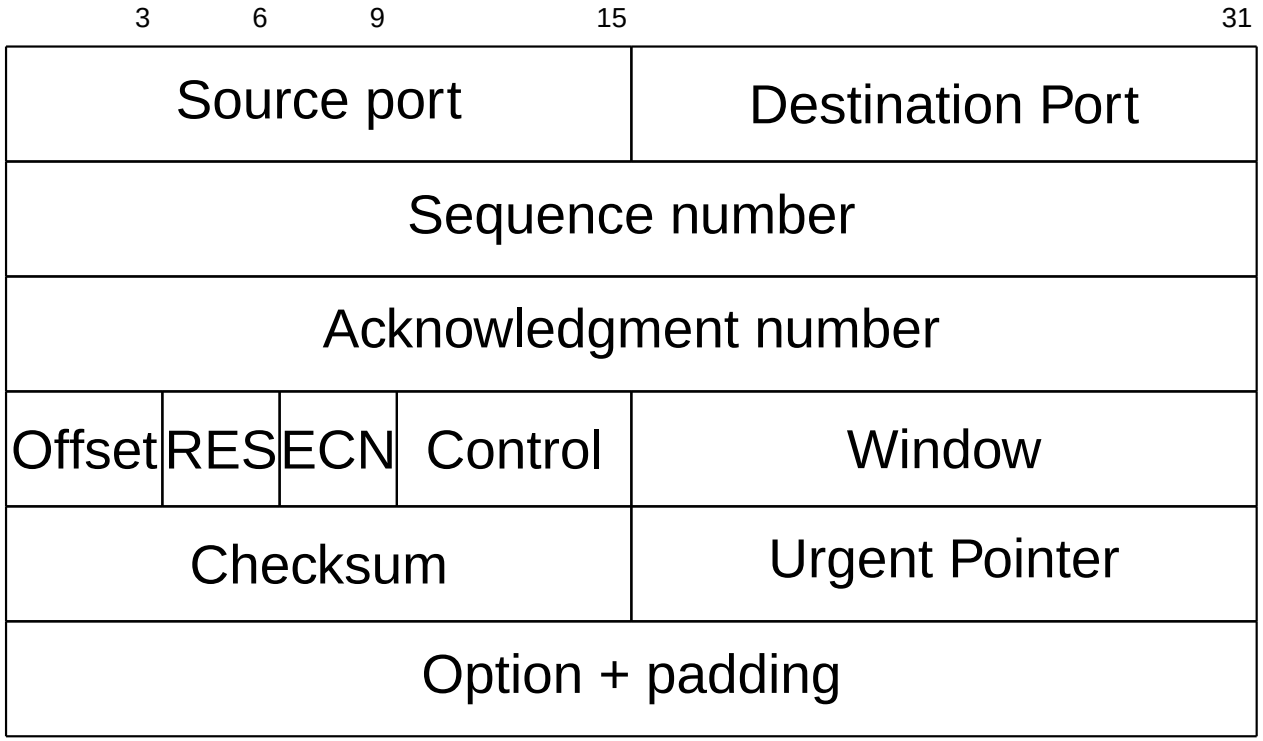


- Il protocollo IP permette di spezzare pacchetti in frammenti più piccoli di un header TCP. Inoltre i pacchetti che arrivano successivamente possono riscrivere quelli arrivati in precedenza.

Fragmentation attacks, segue

Overlapping fragments attack

- **TCP Packet Header:**

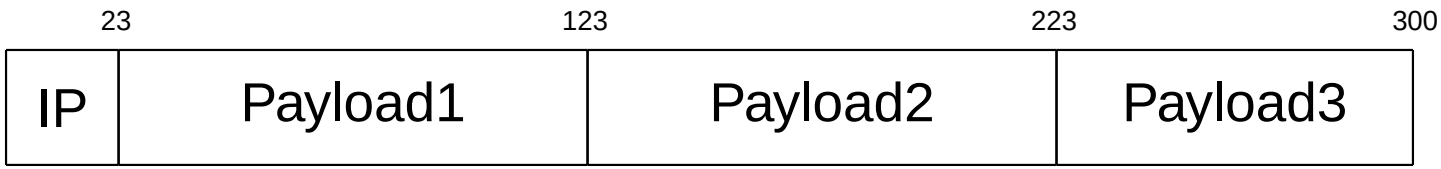


- Il protocollo IP permette di spezzare pacchetti in frammenti più piccoli di un header TCP. Inoltre i pacchetti che arrivano successivamente possono riscrivere quelli arrivati in precedenza.

Fragmentation attacks, segue

Condizioni normali:

- Pacchetto IP di partenza:



- Primo frammento:

- Secondo frammento:

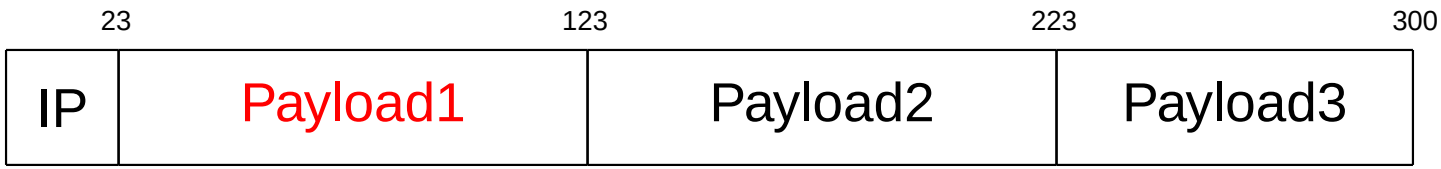
- Terzo frammento:

Leonardo Maccari,
maccari@lenst.det.unifi.

Fragmentation attacks, segue

Condizioni normali:

- Pacchetto IP di partenza:



- Primo frammento:

- Secondo frammento:

- Terzo frammento:

Leonardo Maccari,
maccari@lenst.det.unifi.

Fragmentation attacks, segue

Condizioni normali:

- Pacchetto IP di partenza:

- Primo frammento:

- Secondo frammento:

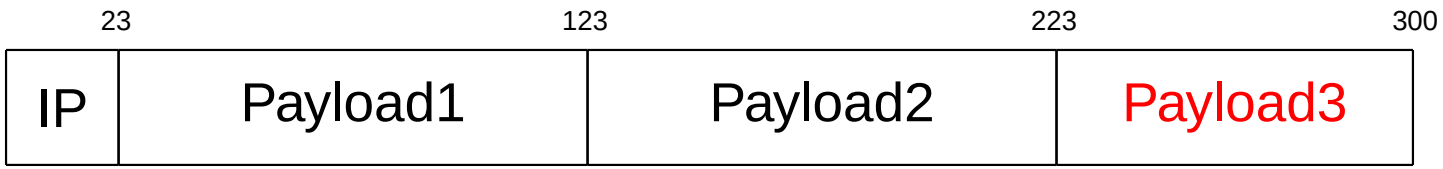
- Terzo frammento:

Leonardo Maccari,
maccari@lenst.det.unifi.

Fragmentation attacks, segue

Condizioni normali:

- Pacchetto IP di partenza:



- Primo frammento:

- Secondo frammento:

- Terzo frammento:

Leonardo Maccari,
maccari@lenst.det.unifi.

Fragmentation attacks, segue

Tiny fragments attack:

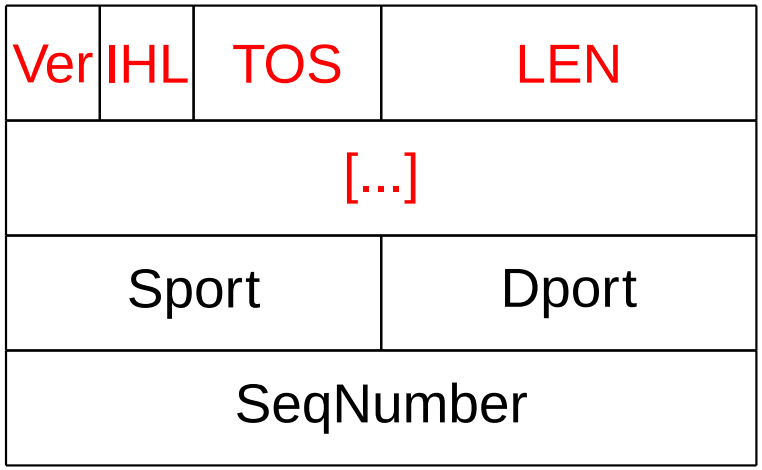
- Pacchetto IP di partenza:

Fragmentation attacks, segue

Tiny fragments attack:

- Pacchetto IP di partenza:

- Primo frammento (8 byte):



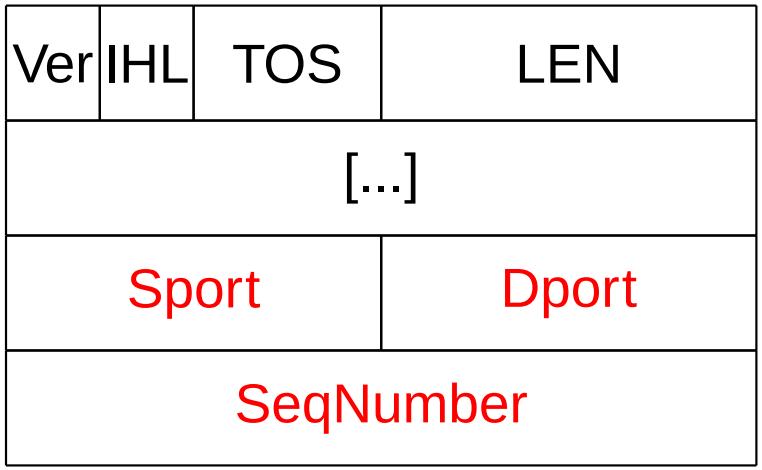
Header IP

Fragmentation attacks, segue

Tiny fragments attack:

- Pacchetto IP di partenza:

- Primo frammento (8 byte):



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:

- 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.

Leonardo Maccari,
maccari@lenst.det.unifi.

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:

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

Leonardo Maccari,
maccari@lenst.det.unifi.

Fragmentation attacks, segue:

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

- Primo frammento

- Secondo frammento

Leonardo Maccari,
maccari@lenst.det.unifi.

Fragmentation attacks, segue:

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

- Primo frammento

- Secondo frammento

Leonardo Maccari,
maccari@lenst.det.unifi.

Fragmentation attacks, segue:

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

- Primo frammento

- Secondo frammento

Leonardo Maccari,
maccari@lenst.det.unifi.

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.

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.

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.

Leonardo Maccari,
maccari@lenst.det.unifi.

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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 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 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 [4]:
  - 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 riceve il terzo pacchetto della connessione, questo contiene l'ACK inviato, da cui si riestraggono le informazioni codificate. A quel punto la connessione è aperta.
  - Standard compliant?

Leonardo Maccari,
maccari@lenst.det.unifi.

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 [4]:
  - 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.
  - Standard compliant?

Leonardo Maccari,
maccari@lenst.det.unifi.

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 [4]:
  - 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.
  - Standard compliant?

Leonardo Maccari,
maccari@lenst.det.unifi.

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 [4]:
  - 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.
  - Standard compliant?

Leonardo Maccari,
maccari@lenst.det.unifi.

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 [4]:
  - 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.
  - Standard compliant?

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*.

Leonardo Maccari,
maccari@lenst.det.unifi.

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*.

Leonardo Maccari,
maccari@lenst.det.unifi.

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*.

Leonardo Maccari,
maccari@lenst.det.unifi.

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*.

Leonardo Maccari,
maccari@lenst.det.unifi.

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*.

Leonardo Maccari,
maccari@lenst.det.unifi.

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.

Leonardo Maccari,
maccari@lenst.det.unifi.

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.

Leonardo Maccari,
maccari@lenst.det.unifi.

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.

Leonardo Maccari,
maccari@lenst.det.unifi.

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.

Leonardo Maccari,
maccari@lenst.det.unifi.

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*:

- 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";

?>
```


Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi al *middleware*:

- 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*:

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

user	password	secret
Alice	123	2131
Bob	321	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'

Attacchi al *middleware*:

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

user	password	secret
Alice	123	2131
Bob	321	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’

Attacchi al *middleware*:

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

user	password	secret
Alice	123	2131
Bob	321	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’

Attacchi al *middleware*:

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

user	password	secret
Alice	123	2131
Bob	321	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’

Attacchi al *middleware*:

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

user	password	secret
Alice	123	2131
Bob	321	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’

Attacchi al *middleware*:

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

user	password	secret
Alice	123	2131
Bob	321	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*:

- 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:

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi al *middleware*:

- 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:

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi al *middleware*:

- 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*:

- 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*:

- 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

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi al *middleware*:

- 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

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi al *middleware*:

- 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

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi al *middleware*:

- 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

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi alle applicazioni

Attacchi da remoto

- buffer overflow
- format bug

Attacchi da locale di elevazione di privilegi

- race condition
- problemi della chiamata system()

Leonardo Maccari,
maccari@lenst.det.unifi.

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 applicazioni

Attacchi da remoto

- buffer overflow
- format bug

Attacchi da locale di elevazione di privilegi

- race condition
- problemi della chiamata system()

Leonardo Maccari,
maccari@lenst.det.unifi.

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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 è: %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()**

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi alle applicazioni:

Buffer overflow

Consideriamo un programma in C come questo:

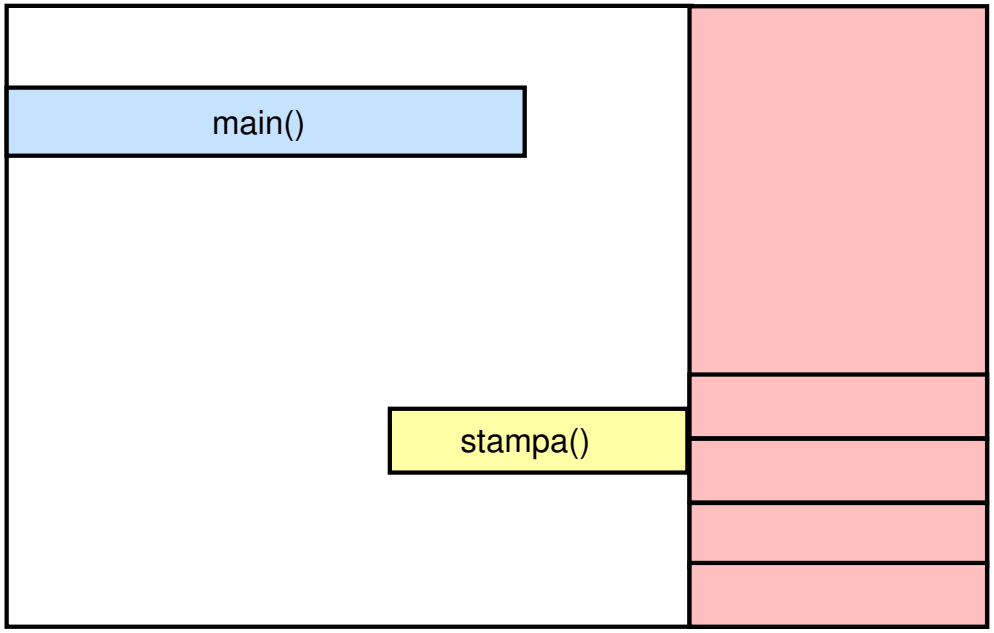
```
#include <stdio.h>
#include <string.h>

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 è: %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()**

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi alle applicazioni:

Buffer overflow

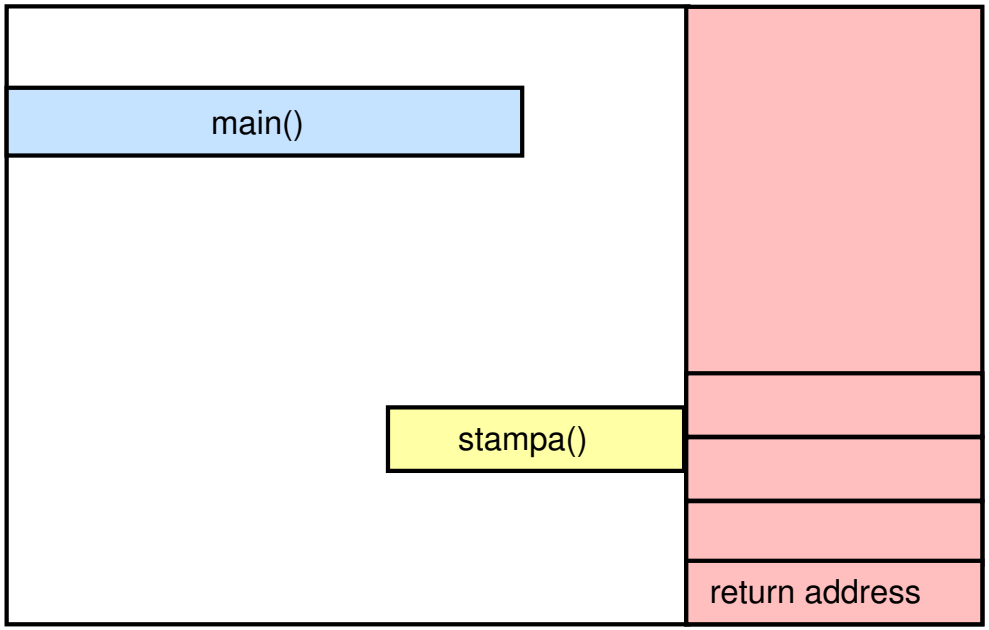


- I passaggi di parametri avvengono mettendo le variabili in una zona comune, lo *stack*. Nello stack finisce:
  - L'indirizzo di ritorno a cui **stampa()** deve fare la **jump** quando finisce
  - la variabile **parola**
  - nello stack viene allocata anche la variabile **testo** dalla funzione **stampa()**

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi alle applicazioni:

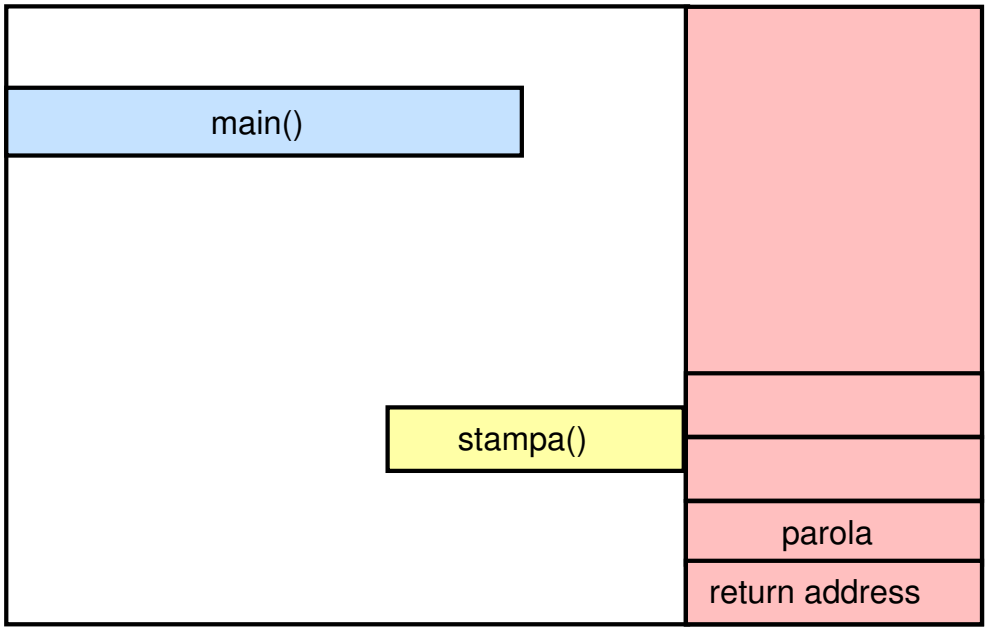
Buffer overflow



- 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:

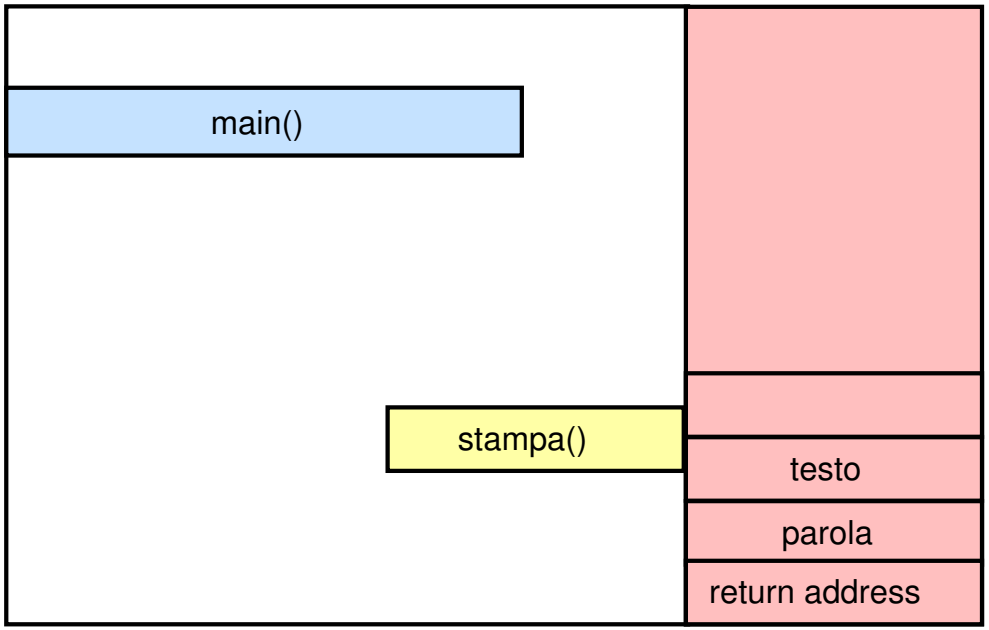
Buffer overflow



- 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()**

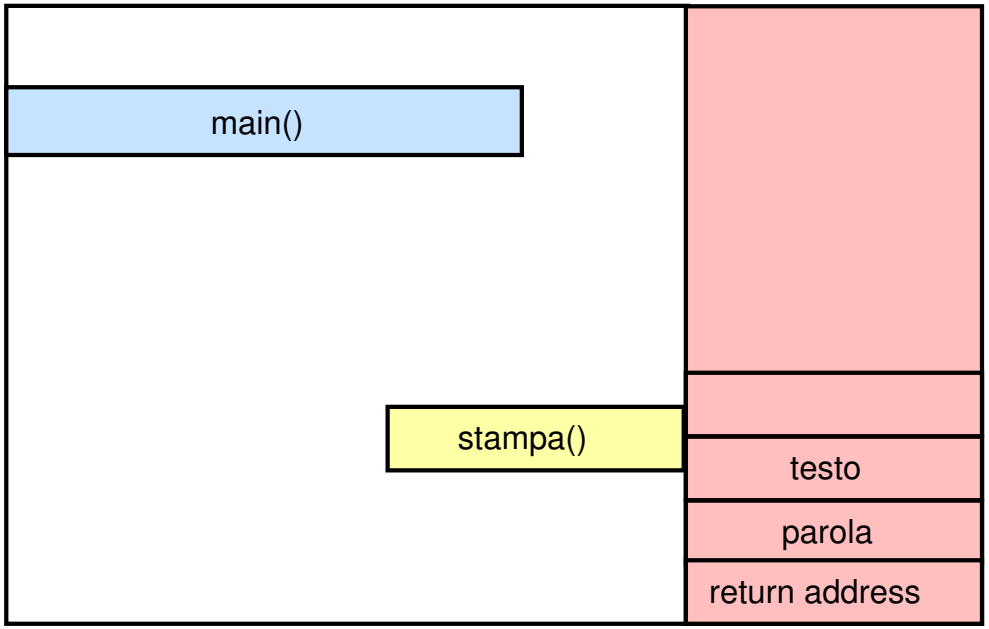
Attacchi alle applicazioni:

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()**

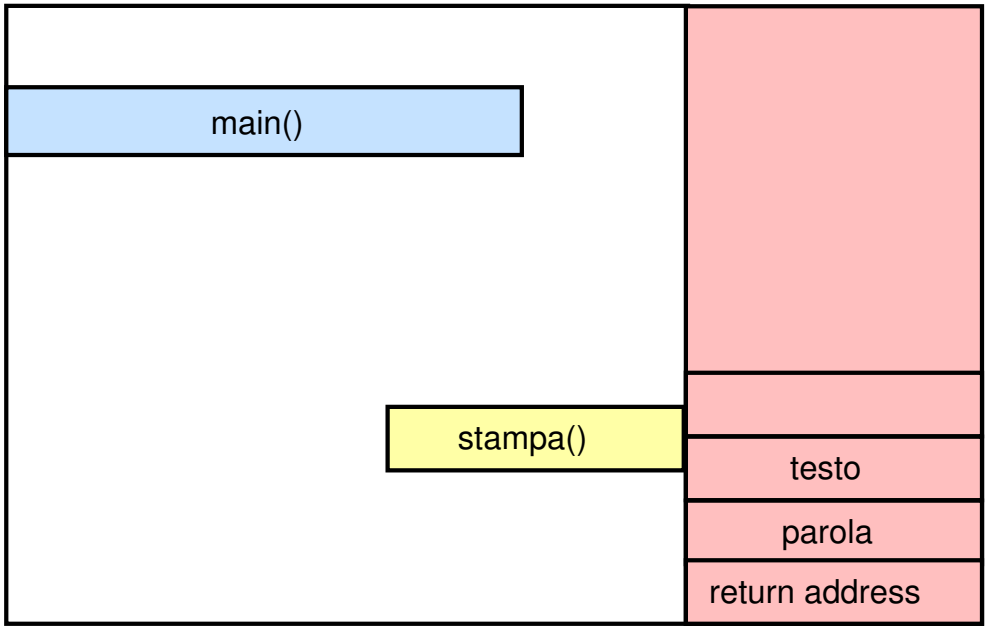
Attacchi alle applicazioni:



- 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!

Leonardo Maccari,
maccari@lenst.det.unifi.

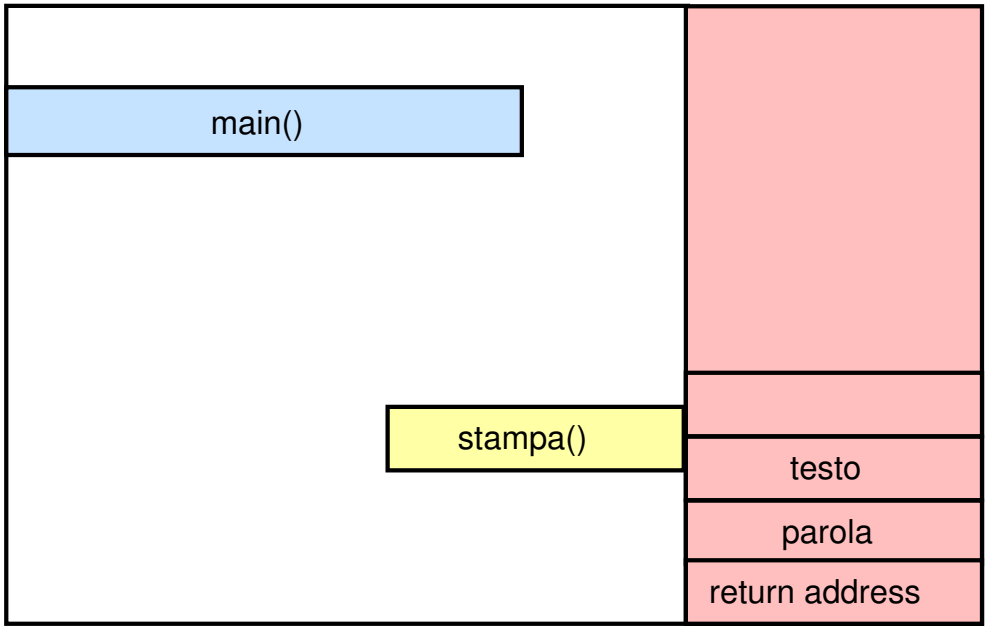
Attacchi alle applicazioni:



- 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!

Leonardo Maccari,
maccari@lenst.det.unifi.

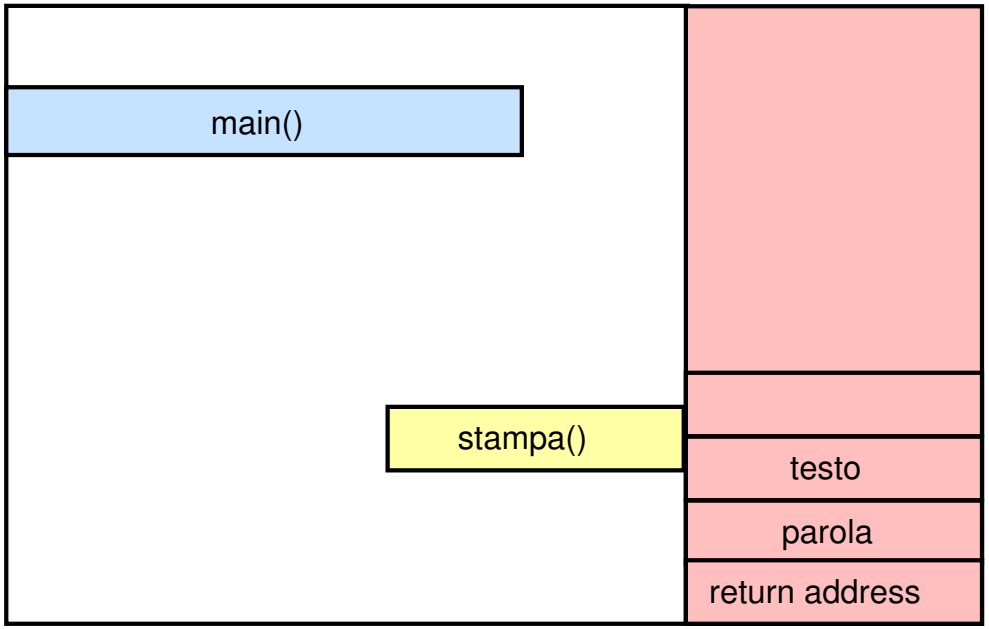
Attacchi alle applicazioni:



- 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!

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi alle applicazioni:



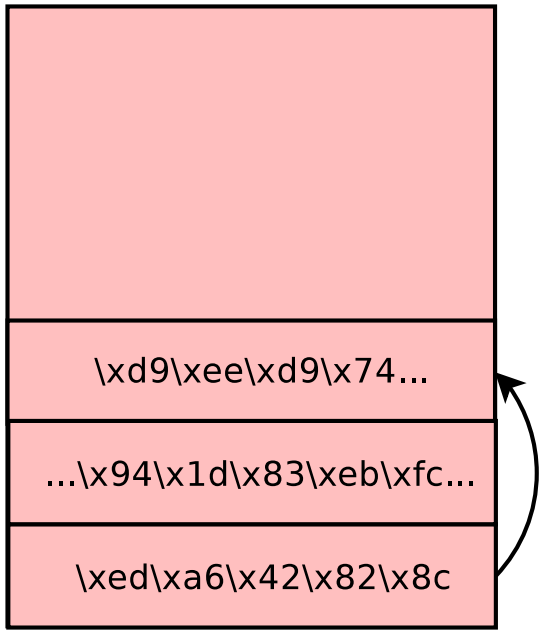
- Nel codice sorgente non si controlla che la variabile **parola** sia lunga meno di 10 char. Se la variabile fosse più lunga dello spazio assegnato andrebbe a sovrascrivere altre zone dello stack, provocando un *segmentation fault*.
  - NB: le variabili nello stack vengono scritte dal basso verso l'alto, 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!

Leonardo Maccari,
maccari@lenst.det.unifi.

Attacchi alle applicazioni:

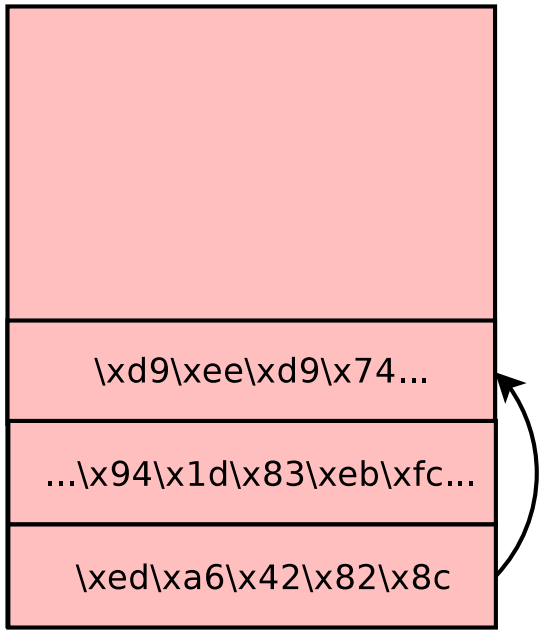
- l'attaccante può scrivere del codice eseguibile nello spazio di memoria della variabile **testo**
- poi far puntare l'indirizzo di ritorno a quel codice
- quando **stampa()** finisce viene fatto un **jump** verso l'indirizzo di ritorno modificato
- il risultato è che si esegue del codice deciso dall'attaccante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

Attacchi alle applicazioni:



- 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:



- 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:

- 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

Leonardo Maccari,
maccari@lenst.det.unifi.

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

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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

Leonardo Maccari,
maccari@lenst.det.unifi.

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:[9]

Leonardo Maccari,
maccari@lenst.det.unifi.

Buone pratiche

- proteggersi dai buffer overflow
- files tempranei
- race condition
- problemi della chiamata system()
- il consiglio per ogni programmatore è di leggersi questo:[9]

Leonardo Maccari,
maccari@lenst.det.unifi.

Buone pratiche

- proteggersi dai buffer overflow
- files tempranei
- race condition
- problemi della chiamata system()
- il consiglio per ogni programmatore è di leggersi questo:[9]

Leonardo Maccari,
maccari@lenst.det.unifi.

Buone pratiche

- proteggersi dai buffer overflow
- files tempranei
- race condition
- problemi della chiamata system()
- il consiglio per ogni programmatore è di leggersi questo:[9]

Leonardo Maccari,
maccari@lenst.det.unifi.

Buone pratiche

- proteggersi dai buffer overflow
- files tempranei
- race condition
- problemi della chiamata system()
- il consiglio per ogni programmatore è di leggersi questo:[9]

Leonardo Maccari,
maccari@lenst.det.unifi.

Programmi con licenze proprietarie

- permettono di utilizzare il programma
- non permettono di vedere o modificare il codice sorgente

Sicurezza

- - Non si sa cosa faccia il programma (*security through obscurity*)
- - Non si può migliorare/patchare, bisogna aspettare le patch
- - Obbediscono soltanto a logiche di mercato (spesso in situazioni di monopolio)
- + Esiste un team di sviluppatori che ci lavora costantemente (davvero?)

"*There's no evidence that any source code has been modified or corrupted*"

(Comunicato stampa Microsoft (10/2000))

Leonardo Maccari,
maccari@lenst.det.unifi.

Programmi con licenze proprietarie

- permettono di utilizzare il programma
- non permettono di vedere o modificare il codice sorgente

Sicurezza

- \- Non si sa cosa faccia il programma (*security through obscurity*)
- \- Non si può migliorare/patchare, bisogna aspettare le patch
- \- Obbediscono soltanto a logiche di mercato (spesso in situazioni di monopolio)
- \+ Esiste un team di sviluppatori che ci lavora costantemente (davvero?)

"*There's no evidence that any source code has been modified or corrupted*"

(Comunicato stampa Microsoft (10/2000))

Programmi con licenze proprietarie

- permettono di utilizzare il programma
- non permettono di vedere o modificare il codice sorgente

Sicurezza

- - Non si sa cosa faccia il programma (*security through obscurity*)
- - Non si può migliorare/patchare, bisogna aspettare le patch
- - Obbediscono soltanto a logiche di mercato (spesso in situazioni di monopolio)
- + Esiste un team di sviluppatori che ci lavora costantemente (davvero?)

"*There's no evidence that any source code has been modified or corrupted*"

(Comunicato stampa Microsoft (10/2000))

Programmi con licenze proprietarie

- permettono di utilizzare il programma
- non permettono di vedere o modificare il codice sorgente

Sicurezza

- - Non si sa cosa faccia il programma (*security through obscurity*)
- - Non si può migliorare/patchare, bisogna aspettare le patch
- - Obbediscono soltanto a logiche di mercato (spesso in situazioni di monopolio)
- + Esiste un team di sviluppatori che ci lavora costantemente (davvero?)

"*There's no evidence that any source code has been modified or corrupted*"

(Comunicato stampa Microsoft (10/2000))

Programmi con licenze proprietarie

- permettono di utilizzare il programma
- non permettono di vedere o modificare il codice sorgente

Sicurezza

- - Non si sa cosa faccia il programma (*security through obscurity*)
- - Non si può migliorare/patchare, bisogna aspettare le patch
- - Obbediscono soltanto a logiche di mercato (spesso in situazioni di monopolio)
- + Esiste un team di sviluppatori che ci lavora costantemente (davvero?)

"There's no evidence that any source code has been modified or corrupted"

(Comunicato stampa Microsoft (10/2000))

Programmi con licenze proprietarie

- permettono di utilizzare il programma
- non permettono di vedere o modificare il codice sorgente

Sicurezza

- - Non si sa cosa faccia il programma (*security through obscurity*)
- - Non si può migliorare/patchare, bisogna aspettare le patch
- - Obbediscono soltanto a logiche di mercato (spesso in situazioni di monopolio)
- + Esiste un team di sviluppatori che ci lavora costantemente (davvero?)

"*There's no evidence that any source code has been modified or corrupted*"

(Comunicato stampa Microsoft (10/2000))

Programmi con licenze libere/open source

- permettono di utilizzare il programma
- permettono di vedere, modificare, ridistribuire il codice sorgente

Sicurezza

- + si sa esattamente cosa fa il programma
  - *Download signed ActiveX backdoor*
  - *Netscape engineers are weenies!*
- + si può migliorare/patchare, non bisogna aspettare le patch
- + il codice viene riutilizzato (anche dagli altri), quindi migliora
- - raramente esiste un team di sviluppatori pagati che progetta e implementa il software

Leonardo Maccari,
maccari@lenst.det.unifi.

Programmi con licenze libere/open source

- permettono di utilizzare il programma
- permettono di vedere, modificare, ridistribuire il codice sorgente

Sicurezza

- + si sa esattamente cosa fa il programma
  - *Download signed ActiveX backdoor*
  - *Netscape engineers are weenies!*
- + si può migliorare/patchare, non bisogna aspettare le patch
- + il codice viene riutilizzato (anche dagli altri), quindi migliora
- - raramente esiste un team di sviluppatori pagati che progetta e implementa il software

Programmi con licenze libere/open source

- permettono di utilizzare il programma
- permettono di vedere, modificare, ridistribuire il codice sorgente

Sicurezza

- + si sa esattamente cosa fa il programma
  - *Download signed ActiveX backdoor*
  - *Netscape engineers are weenies!*
- + si può migliorare/patchare, non bisogna aspettare le patch
- + il codice viene riutilizzato (anche dagli altri), quindi migliora
- - raramente esiste un team di sviluppatori pagati che progetta e implementa il software

Programmi con licenze libere/open source

- permettono di utilizzare il programma
- permettono di vedere, modificare, ridistribuire il codice sorgente

Sicurezza

- + si sa esattamente cosa fa il programma
  - *Download signed ActiveX backdoor*
  - *Netscape engineers are weenies!*
- + si può migliorare/patchare, non bisogna aspettare le patch
- + il codice viene riutilizzato (anche dagli altri), quindi migliora
- - raramente esiste un team di sviluppatori pagati che progetta e implementa il software

Programmi con licenze libere/open source

- permettono di utilizzare il programma
- permettono di vedere, modificare, ridistribuire il codice sorgente

Sicurezza

- + si sa esattamente cosa fa il programma
  - *Download signed ActiveX backdoor*
  - *Netscape engineers are weenies!*
- + si può migliorare/patchare, non bisogna aspettare le patch
- + il codice viene riutilizzato (anche dagli altri), quindi migliora
- - raramente esiste un team di sviluppatori pagati che progetta e implementa il software

Programmi con licenze libere/open source

- permettono di utilizzare il programma
- permettono di vedere, modificare, ridistribuire il codice sorgente

Sicurezza

- + si sa esattamente cosa fa il programma
  - *Download signed ActiveX backdoor*
  - *Netscape engineers are weenies!*
- + si può migliorare/patchare, non bisogna aspettare le patch
- + il codice viene riutilizzato (anche dagli altri), quindi migliora
- - raramente esiste un team di sviluppatori pagati che progetta e implementa il software

Programmi con licenze libere/open source

- permettono di utilizzare il programma
- permettono di vedere, modificare, ridistribuire il codice sorgente

Sicurezza

- + si sa esattamente cosa fa il programma
  - *Download signed ActiveX backdoor*
  - *Netscape engineers are weenies!*
- + si può migliorare/patchare, non bisogna aspettare le patch
- + il codice viene riutilizzato (anche dagli altri), quindi migliora
- - raramente esiste un team di sviluppatori pagati che progetta e implementa il software

Programmi con licenze libere/open source

- permettono di utilizzare il programma
- permettono di vedere, modificare, ridistribuire il codice sorgente

Sicurezza

- + si sa esattamente cosa fa il programma
  - *Download signed ActiveX backdoor*
  - *Netscape engineers are weenies!*
- + si può migliorare/patchare, non bisogna aspettare le patch
- + il codice viene riutilizzato (anche dagli altri), quindi migliora
- - raramente esiste un team di sviluppatori pagati che progetta e implementa il software

RFC 826: http://www.faqs.org/rfcs/rfc826.html

http://packetstormsecurity.com/papers/protocols/intro_to_arp_spoofing.pdf

CERT advisory
http://www.cert.org/advisories/CA-1996-21.html

http://cr.yp.to/syncookies.html

NISCC advisory, esclusa la parte relativa ai vendor:
http://www.uniras.gov.uk/vuls/2004/236929/index.htm

http://www.cert.org/archive/pdf/cross_site_scripting.pdf

fino al capitolo 3.1 di:
http://www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf

Una semplice introduzione: http://www.linuxjournal.com/article/6701

http://www.dwheeler.com/secure-programs/

ettercap, programma per realizzare vari attacchi di tipo MITM, http://ettercap.sourceforge.net/

Leonardo Maccari, maccari@lenst.det.unifi.