ESERCITAZIONE FINALE
(21 gennaio 2026)

Corso: **3770/10840604-011/606/DEC/25**
Titolo: **ESPERTO IN SICUREZZA INFORMATICA – ED. ROVIGO**
Sede: **ROVIGO (RO), Via N. Badaloni 2**

Modulo 3: **MONITORAGGIO DELLA SICUREZZA DEL SISTEMA INFORMATIVO**

Docente: Davide Gessi

Corsista: Massimo Zeri Guerra

Valutazione: 10/10

Securing Software Esercitazione

1. **Attraverso quali tecniche un attore malevolo potrebbe “craccare” un software, cioè bypassare la registrazione o il pagamento per poterlo usare gratuitamente?**
2. **Distingui la natura dei due tipi di attacchi “cross-site” discussi:**
   - Cross-Site Scripting (XSS)
   - Cross-Site Request Forgery (CSRF)
3. **Perché è necessario “fare l'escape” (cioè neutralizzare) alcuni caratteri nei dati di input?**
4. **Nel contesto di SQL, che cos’è una prepared statement (istruzione preparata)?**
5. **Perché la validazione lato client (client-side validation) è considerata meno sicura rispetto a quella lato server (server-side validation)?**
6. **Riferimento alla vignetta GeekHero**
   Dal punto di vista pratico, la vignetta sopra ha probabilmente ragione nel rappresentare il comportamento della maggior parte degli utenti verso il software open-source.
   Tuttavia, anche se questo fosse la tua opinione, perché potrebbe comunque essere una buona idea usare (o sviluppare) più software open-source, dal punto di vista della sicurezza informatica?
7. **In che modo i package manager (come apt, yum, npm, ecc.) sono simili agli app store (Apple App Store, Google Play Store, Microsoft Store, ecc.) dal punto di vista della cybersecurity?**
8. **Contro quale tipo di minaccia aiuta a difendersi l’uso del campo Content-Security-Policy (CSP) nel nostro codice sorgente?**
9. **Fornisci un esempio concreto di una situazione in cui potresti voler usare il metodo HTTP POST invece del metodo GET.**
10. **Heartbleed (CVE-2014-0160)**
    Il bug noto come Heartbleed, scoperto nel 2014, generò un’enorme preoccupazione su Internet: fu uno dei primi casi in cui una vulnerabilità informatica venne diffusa anche dai media generalisti, mentre i ricercatori di sicurezza cercavano di avvisare il pubblico e incoraggiare un aggiornamento urgente dei sistemi.
    Leggi informazioni su Heartbleed, ad esempio dalla pagina Wikipedia o da altre fonti affidabili (come un video divulgativo).
    Perché Heartbleed rappresentava una minaccia così grave per la sicurezza degli utenti?
    Qua la vignetta obbligatoria xkcd

Risposte:

1)Attraverso quali tecniche un attore malevolo potrebbe “craccare” un software, cioè bypassare la registrazione
o il pagamento per poterlo usare gratuitamente?

R)

Attraverso tecniche :
Reverse engineering, debugging, patching binario, keygen, hooking, emulazione licenza, tampering memoria.

Le tecniche principali sono: reverse engineering per analizzare il codice con disassemblatori/debugger e trovare i controlli di licenza (es. if registrato), patching per modificare istruzioni binarie e bypassare i check, keygen per ricreare l’algoritmo di validazione delle seriali, hooking per intercettare API e mentire sul software, emulazione per fingere chiavi hardware/file, tampering per alterare variabili runtime come flag licensed
Eccellente. lessico da addetto ai lavori.

2) Distingui la natura dei due tipi di attacchi “cross-site” discussi:
- Cross-Site Scripting (XSS)
- Cross-Site Request Forgery (CSRF)

R)

XSS: inietta ed esegue codice JS nel browser della vittima (es. via input non escaped in cerca.php), sfruttando il contesto
della sessione per azioni malevole (cambio password, worm). CSRF: forza richieste non autorizzate dal browser della vittima
autenticata verso un altro sito (es. img src o form auto-submit), usando i suoi cookie senza eseguire codice sul target.

XSS (Reflected/Stored) è code injection client-side: input utente (es. <script>alert('attacco')</ interpretato/eseguito come HTML/JS nella pagina vittima, offuscato con btoa/urlencode per link phishing. CSRF è request
forgery: sfrutta autenticazione esistente per azioni indesiderate (es. compra via GET in <img src> o POST auto-submit),
prevenuto con token CSRF.
Molto buono. Conosci il XSS/CSRF anche oltre il minimo richiesto.

3) Perché è necessario “fare l'escape” (cioè neutralizzare) alcuni caratteri nei dati di input?

R) L'escaping neutralizza caratteri speciali negli input utente (es. <, >, &, ", ') per prevenire iniezioni malevole come
XSS o SQLi, trasformandoli in entità sicure (es. <, >). Senza, un input come <script>alert('attacco')</
eseguito come codice JS invece di testo visualizzato.
Prevenzione XSS: Caratteri HTML/JS (<script>) chiudono tag prematuramente, iniettando codice eseguibile nel browser vittima.
Protezione SQL/Command Injection: Apici o ; alterano query (es. ' OR 1=1; DROP TABLE), eseguendo comandi non autorizzati.
Output Sicuro: Prima di emettere input in HTML/PHP/SQL, escape rende innocui (es. <p><script>alert('attacco')</

Esempi di Escaping:

Contesto	Caratteri Pericolosi	Escape Esempio
HTML	< > & " '	< > & " ' Securing-Software.htm
SQL	' ; --	'' o prepared statements Securing-Software.htm
Shell	; ` $	escapeshellarg() [file:235]

Corretto. Concetto giusto
4) Nel contesto di SQL, che cos'è una prepared statement (istruzione preparata)?

R) Una prepared statement è un template SQL precompilato con segnaposto (?) per parametri utente, separando struttura query
da dati per prevenire SQL injection tramite escaping automatico (es. raddoppiamento apici). Parametri vengono bound
separatamente, trattando input come dati literals anziché codice eseguibile.

Come Funziona:

Preparazione: Query fissa con placeholder, es. SELECT * FROM users WHERE username = ? AND password = ?.
Binding: Input utente passati come parametri sicuri, es. username="davide" → 'davide' (escaped).
Esecuzione: DB interpreta solo struttura; input non altera logica (es. ' OR 1=1 diventa harmless).

Esempi di SQL Injection Bloccata :

Query Vulnerabile	Input Malevolo	Risultato
SELECT * FROM users WHERE username = '$user'	davide' DELETE FROM users --	Esegue DROP Securing-Software.htm
Prepared: SELECT * FROM users WHERE username = ?	davide' DELETE FROM users --	Cerca utente letterale, no DROP Securing-Software.htm

Corretto. Concetto giusto

5) Perché la validazione lato client (client-side validation) è considerata meno sicura rispetto a quella lato server
(server-side validation)?

R) La validazione client-side è meno sicura perché eseguita nel browser utente (es. JS o attributi HTML come disabled/required),
facilmente bypassabile con DevTools (Shift+Ctrl+I) modificando codice o inviando richieste dirette. Server-side è affidabile:
controlla dati sul server fidato, non manipolabile da client, replicando regole front-end per sicurezza.

Spiegazioni Salienti Keys point :

Bypass Facile: Utente rimuove disabled da checkbox sconto o required da input, inviando form valido server-side.

Non Fidarsi Client: "Non fidarsi di quello che inserisce un utente; server deve controllare sempre SERVER side".

UX vs Sicurezza: Client-side migliora esperienza (feedback immediato), ma server-side protegge da abusi (es. form shop).

Confronto :

Tipo	Pro	Contro	Esempio Bypass
Client	Veloce, UX buona	Bypassabile	Rimuovi disabled DevTools Securing-Software.htm
Server	Sicura, autorevole	Più lenta	Non bypassabile Securing-Software.htm

Ottimo. Chiaro, centrato,

6) Riferimento alla vignetta GeekHero
Dal punto di vista pratico, la vignetta sopra ha probabilmente ragione nel rappresentare il comportamento della maggior
parte degli utenti verso il software open-source.Tuttavia, anche se questo fosse la tua opinione, perché potrebbe comunque
essere una buona idea usare (o sviluppare) più software open-source, dal punto di vista della sicurezza informatica?

R) Anche se molti utenti non ispezionano il codice, l'open-source offre vantaggi di sicurezza grazie alla comunità globale
di esperti che lo fa al posto loro. Questo approccio collettivo identifica e risolve vulnerabilità più rapidamente
rispetto al software closed-source.

Trasparenza Codice

Il codice sorgente pubblico permette audit indipendenti, verificando assenza di backdoor o malware nascosti.
Esperti mondiali lo esaminano, applicando "Linus's Law": con abbastanza occhi, tutti i bug sono superficiali.

Patch Rapide

Vulnerabilità scoperte portano a fix immediati dalla comunità, senza dipendere da un singolo vendor. Progetti
come Linux o Apache beneficiano di aggiornamenti veloci, riducendo tempi di esposizione.

Community Audits

Sviluppo collaborativo include revisioni continue da security researcher e developer. Bug bounty e contributi
globali rafforzano la resilienza, superando ispezioni interne di software proprietari.

Confronto Sicurezza :

Aspetto	Open-Source	Closed-Source
Audit	Globale, rapido negg	Interno, limitato korte
Fix Vulnerabilità	Community immediata horilla	Vendor-dipendente linuxsecurity
Backdoor Rischio	Basso (visibile) negg	Alto (nascosto) linuxsecurity

Buono. Hai fatto un saggio

7) In che modo i package manager (come apt, yum, npm, ecc.) sono simili agli app store (Apple App Store, Google Play Store,
Microsoft Store, ecc.) dal punto di vista della cybersecurity?

R) I package manager come apt, yum e npm sono simili agli app store perché entrambi forniscono repository centralizzati
di software pre-verificati, riducendo rischi da download casuali. Entrambi usano firme digitali per garantire integrità
e autenticità pacchetti, prevenendo manomissioni

Firma Digitale

Package manager (apt, rpm) firmano pacchetti con chiavi GPG, verificando provenienza da fonti ufficiali.
App store (Apple, Google) usano firme per hash software, bloccando installazioni alterate.

Repository Centrali

Offrono update automatici sicuri da canali fidati, minimizzando supply chain attacks. Evitano

download da siti
non verificati, simili a restrizioni app store su sideload.

Controllo Accesso

Limitano installazioni a fonti approvate, gestendo permessi e dipendenze. Trend: OS integrano package
manager come app store per ecosistemi chiusi ma sicuri

Confronto :

Feature	Package Manager	App Store
Firma	GPG keys (apt/rpm) Securing-Software.htm	Private key hash Securing-Software.htm
Update	Automatici sicuri Securing-Software.htm	Centralizzati Securing-Software.htm
Limiti	Repo ufficiali Securing-Software.htm	No sideload Securing-Software.htm

Corretto.

8) Contro quale tipo di minaccia aiuta a difendersi l'uso del campo Content-Security-Policy (CSP) nel nostro codice sorgente?

R) Il campo Content-Security-Policy (CSP) difende principalmente contro Cross-Site Scripting (XSS), bloccando
inline script e risorse esterne non autorizzate. Specifica sorgenti per script, stili e connessioni, prevenendo
esecuzione di codice malevolo iniettato.

Blocco Inline Script

CSP con script-src 'self' permette solo script da file del dominio, bloccando <script>alert('xss')</script> inline
da XSS. Previene eval o caricamenti esterni.

Controllo Risorse

Header come Content-Security-Policy: script-src https://sguaff.com limita a domini fidati, fermando fetch
cross-domain o stili malevoli. Anche connect-src self blocca richieste outbound.

Esempi Protezione

script-src 'self': Solo script locali, no inline.

style-src 'self': Blocca CSS inline/esterne.

vedi Tabaella :

Direttiva CSP	Minaccia Bloccata	Esempio Header Securing-Software.htm
script-src	Inline XSS	script-src httpssguaff.com
connect-src	Fetch cross-site	connect-src self

Molto buono. CSP spiegata bene, anche troppo in dettaglio.

9) Fornisci un esempio concreto di una situazione in cui potresti voler usare il metodo HTTP POST invece del metodo GET.

R) Un esempio concreto è l'acquisto di un prodotto su un e-commerce come Amazon, dove un link GET semplice
può essere abusato in un attacco CSRF tramite un tag img malevolo. Usa POST con un form per nascondere parametri
sensibili come l'ID prodotto nell'URL e richiedere un'azione intenzionale.

Protezione CSRF

GET espone parametri (es. https://amazon.it/dp/), caricabili involontariamente via
<img src="...">causando acquisti non autorizzati. POST nel form richiede submit esplicito:
<form method="post" action="..."><input name="dp" value="B07XLQ2FSK"><button>

Vantaggi POST

Parametri nel body, non URL, prevengono log sensibili e accessi accidentali. Aggiungi CSRF token per extra sicurezza: <input name="csrftoken" value="1234abcd">.

Vedi tabella :

Metodo	Esempio URL/Form	Rischio CSRF Securing-Software.htm
GET	/buy?dp=B07XLQ2FSK	Alto (img tag)
POST	<form method=post>	Basso (richiede click)

Buona. Ma leggermente forzata. POST diverso da protezione CSRF

10) Heartbleed (CVE-2014-0160)
Il bug noto come Heartbleed, scoperto nel 2014, generò un'enorme preoccupazione su Internet: fu uno dei primi casi
in cui una vulnerabilità informatica venne diffusa anche dai media generalisti, mentre i ricercatori di sicurezza
cercavano di avvisare il pubblico e incoraggiare un aggiornamento urgente dei sistemi.Leggi informazioni su Heartbleed,
ad esempio dalla pagina Wikipedia o da altre fonti affidabili (come un video divulgativo).
Perché Heartbleed rappresentava una minaccia così grave per la sicurezza degli utenti?

R) Heartbleed (CVE-2014-0160) era grave perché permetteva a un attaccante remoto di leggere fino a

64KB di memoria
server per richiesta heartbeat TLS, leakando dati sensibili come chiavi private senza privilegi o autenticazione.

Colpiva OpenSSL 1.0.1-1.0.1f, usato dal 17% dei server HTTPS, esponendo password, cookie e certificati per oltre
due anni.

Meccanismo Vulnerabilità

Mancato controllo bounds prima di memcpy() in heartbeat: l'attaccante invia lunghezza falsa (es. 64KB invece di
3 byte), server copia memoria extra e la restituisce. Ripetibile per estrarre dati critici come session keys.

Impatto Massiccio

Esposizione chiavi private: decrittazione traffico passato/futuro, impersonificazione siti.
Dati rubati: credenziali, carte credito, record medici; difficile rilevare.
Vulnerabile 24-55% siti HTTPS; attacco facile con PoC, persistente da 2012.

Conseguenze Utenti

Richiedeva cambio password globali, reissuing certificati (costo centinaia milioni), panico mediatico per breach silenti su VPN/VOIP interni. Persiste su legacy systems.

Aspetto	Rischio Principale	Esempio Conseguenza owasp+1
Accesso	Remoto, anonimo	Leak chiavi private senza login
Dati Esposti	Memoria heap casuale	Password, cookie, certificati
Durata	Oltre 2 anni	Breach retroattivi possibili
Mitigazione	Patch + revoke certs	Cambio password worldwide

Ottimo. quasi da presentazione OWASP. Fuori scala per un quiz