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: Riccardo Berto

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. Un attore malevolo può craccare un software utilizzando tecniche di reverse engineering per analizzare l'eseguibile e comprenderne il funzionamento interno. Tramite disassembler e debugger può individuare i controlli di registrazione o pagamento presenti nel codice. Una volta trovati, tali controlli possono essere aggirati applicando patch binarie che modificano direttamente le istruzioni dell'eseguibile. In alternativa, è possibile sviluppare keygen che ricreano l'algoritmo di generazione delle chiavi di licenza. Altre tecniche includono l'hooking delle API per intercettare e falsificare i risultati dei controlli, l'emulazione di server o dispositivi di licenza e il tampering della memoria a runtime per alterare variabili come i flag di licenza. **Eccellente.** Tutte le principali tecniche di cracking menzionate, spiegazione chiara e ordinata.
2. Il Cross-Site Scripting (XSS) è un attacco in cui un aggressore inietta codice malevolo, tipicamente JavaScript, in una pagina web che viene poi eseguito nel browser della vittima. Questo avviene quando l'input dell'utente non viene correttamente validato o "escaped" e viene interpretato come codice. Il Cross-Site Request Forgery (CSRF), invece, induce il browser di una vittima autenticata a inviare richieste non intenzionali verso un sito web, sfruttando i cookie di sessione già validi. A differenza dell'XSS, nel CSRF non viene eseguito codice sul sito bersaglio, ma si sfrutta la fiducia del server nel browser dell'utente. In sintesi, l'XSS sfrutta la fiducia dell'utente nel sito, mentre il CSRF sfrutta la fiducia del sito nell'utente. **Molto buona.** Differenza XSS e CSRF corretta Concetti chiave chiari
3. È necessario fare l'escape dei caratteri di input perché alcuni caratteri speciali possono essere interpretati come codice anziché come semplici dati. Senza escaping, un input fornito dall'utente potrebbe alterare la struttura di una pagina HTML, di uno script o di una query SQL. Questo può portare ad attacchi di injection, come Cross-Site Scripting o SQL injection. L'escaping trasforma i caratteri pericolosi in una rappresentazione sicura, impedendone l'esecuzione. In questo modo l'input viene trattato esclusivamente come testo e non come istruzioni. L'escape è quindi una misura fondamentale per garantire che i dati dell'utente non compromettano il comportamento dell'applicazione.**Buono.** Concetto corretto di escape, con attenzione a contesto HTML/JS/SQL
4. Una prepared statement è un'istruzione SQL in cui la struttura della query viene definita e compilata separatamente dai dati forniti dall'utente. La query contiene dei segnaposto

per i parametri, che vengono successivamente associati a valori concreti. In questo modo, l'input dell'utente non viene interpretato come parte del codice SQL, ma solo come dato. Questo meccanismo previene efficacemente le SQL injection, poiché i caratteri speciali non possono modificare la logica della query. Inoltre, le prepared statement migliorano l'efficienza quando la stessa query viene eseguita più volte, riducendo il lavoro di parsing del database. **Corretto.** Prepared statement spiegato bene, sia lato sicurezza sia lato efficienza.

5. La validazione lato client è considerata meno sicura perché viene eseguita nel browser dell'utente, che non è un ambiente fidato. Un attaccante può facilmente aggirare questi controlli disabilitando JavaScript o modificando le richieste inviate al server. È possibile inviare dati arbitrari direttamente tramite strumenti come curl o proxy di intercettazione. La validazione lato server, invece, avviene in un ambiente controllato dall'applicazione e non può essere manipolata dall'utente. Per questo motivo, ogni dato ricevuto dal client deve essere sempre validato nuovamente sul server. La validazione client-side può migliorare l'esperienza utente, ma non deve mai essere considerata una misura di sicurezza sufficiente. **Molto buono.** Spiegato chiaramente.
6. Anche se molti utenti non analizzano direttamente il codice sorgente, il software open-source offre importanti vantaggi di sicurezza. Il codice pubblico può essere esaminato da una vasta comunità di sviluppatori ed esperti di sicurezza, aumentando la probabilità che vulnerabilità e bug vengano individuati. Questo approccio riduce la dipendenza dal "security through obscurity", che non garantisce una reale protezione. Inoltre, le vulnerabilità scoperte possono essere corrette rapidamente dalla comunità senza attendere un singolo vendor. La trasparenza del codice consente anche di verificare l'assenza di backdoor o comportamenti malevoli. Nel complesso, l'open-source favorisce una maggiore robustezza e affidabilità dei sistemi. **Molto buono.** Vantaggi open-source ben spiegati
7. Dal punto di vista della cybersecurity, i package manager e gli app store svolgono un ruolo simile nella distribuzione del software. Entrambi forniscono repository centralizzati da cui gli utenti scaricano applicazioni provenienti da fonti considerate affidabili. Utilizzano meccanismi di verifica dell'integrità e dell'autenticità, come le firme digitali, per prevenire la manomissione dei pacchetti. Inoltre, consentono aggiornamenti automatici che includono patch di sicurezza. Questo modello riduce il rischio di installare software malevolo da fonti non controllate. Tuttavia, una compromissione del repository o dello

store può avere un impatto su un grande numero di utenti, rendendo critica la sicurezza della catena di distribuzione. **Buono.** Confronto tra package manager e app store chiaro

8. La Content-Security-Policy (CSP) aiuta a difendersi principalmente da attacchi di Cross-Site Scripting (XSS). Attraverso specifiche direttive, la CSP limita le sorgenti da cui il browser può caricare ed eseguire script, stili e altre risorse. In questo modo viene impedita l'esecuzione di codice inline o proveniente da domini non autorizzati. Anche in presenza di una vulnerabilità di injection, la CSP può ridurre o annullare l'impatto dell'attacco. La CSP rappresenta quindi una difesa aggiuntiva lato browser contro l'esecuzione di codice malevolo. **Corretto.** CSP spiegata con precisione
9. Un esempio concreto in cui è preferibile usare il metodo HTTP POST è l'invio di dati sensibili tramite un form, come nel caso di un acquisto o di un login. Con il metodo GET, i parametri vengono inseriti nell'URL e possono essere salvati nella cronologia del browser o nei log del server. Inoltre, una richiesta GET può essere facilmente attivata in modo involontario, ad esempio tramite un tag img, facilitando attacchi CSRF. Il metodo POST invia invece i dati nel corpo della richiesta, riducendo la visibilità dei parametri. L'uso di POST richiede un'azione più esplicita dell'utente e consente l'integrazione di meccanismi di protezione come i token CSRF. **Buono.** POST spiegato bene, con cenno a CSRF
10. Heartbleed era una vulnerabilità critica perché permetteva a un attaccante remoto di leggere porzioni arbitrarie della memoria di un server che utilizzava OpenSSL. Questo avveniva senza necessità di autenticazione e senza lasciare tracce evidenti. Attraverso questo bug era possibile ottenere informazioni estremamente sensibili, come password, cookie di sessione e persino chiavi private TLS. Poiché OpenSSL era ampiamente diffuso, la vulnerabilità colpiva una vasta parte dell'infrastruttura HTTPS globale. La compromissione delle chiavi private rendeva inoltre inutili le comunicazioni cifrate, minando la fiducia nella sicurezza di Internet. **Perfetto.** Heartbleed spiegato correttamente