SecureBlog

Secure Development Lifecycle

David Pontes | up202209310@fc.up.pt

Lourenço Antunes | up202203893@fc.up.pt

Luís Oliveira | up201108115@fc.up.pt

Table of contents

Executive Summary

We have conducted a security assessment of our secure blogging application – *SecureBlog* – in order to assess the security qualities of the implementation, determine existing vulnerabilities, fragile dependencies and establish a security risk associated with the system.

The assessment was conducted in accordance with the OWASP Web Security Testing Guide, version 4.2 and harnessed threat modeling, static code analysis and code review, penetration testing and static code analysis.

Some of the findings uncovered while performing were remediated during the development phase.

Scope

Environment

All tests were performed using a development environment.

Limitations

- Production environment
  - Security issues related to deployment and deployment environment(s) were not tested. In a real setting, this would be a very important factor to take into consideration when performing security tests.
- Social engineering
  - Social engineering attacks or exploits were not in scope for this assessment.
- In-house attacks
  - Only client side attacks were considered for this assessment and the internal SecDevOps company perimeter was not considered.
- Third-party software and code
  - Only python code is considered. Additional dependencies required for displaying web pages, i.e., css or javascript, are not considered.

Provided user accounts

Testers were provided access to an administrative account, a content creator account and a regular user account, as detailed in the table below:

Asset	Account	Access Level
Web application	admin	Admin
Web application	SecDevOps	Author
Web application	Fan	User

Tools

Testing was performed using BurpSuite (Community Edition), OWASP ZAP, SonarCloud, bandit, safety and Nessus.

Methodology

Our secure development and security assessment methodologies are based on the ***OWASP Web Security Testing Guide***.

Our test guide covers three of key aspects of a SDLC (*Software Development Life Cycle)*, which are:

1. Testing during all stages of the Life Cycle, namely the ones that are applied in this project:
    a. Before development begins
    b. Design phase
    c. Development phase
2. The second aspect is defining the different approaches to be used in order to test security of the application;
3. Finally, the third aspect is about security features and threat prevention coverage to be achieved and assured by tests by the end of each phase..

Based on these 3 dimensions, the defined test methodology applied to the project can be summarized in three different stages:

1. Before development
    a. Clarification of general principles and assumptions for during all phases
        i. By providing a software architecture for our application
    b. The target results intended for the testing findings and security acceptance criteria
        i. Through security requirements defined in the architecture
        ii. By defining quality goals in the software architecture
    c. Defining and preparing a suitable development environment enabling continuous security testing
        i. A similar development environment was setup for all project members
2. During design
    a. **Review architecture** and business requirements, in order to have clear context for the security requirements;
    b. reviewing security requirements, based on both

(1) Blog security requirements, derived from:

- Functional requirements and key assets.
- Component dependencies
- Abuses cases
- Threat model,

(2) Test in accordance with OWASP's testing guide main categories of tests

i. reviewing design and iterate over item (a) until the design is considered compliant with defined requirements

3. During development

   In this phase, the test guidance we applied was defined in accordance with the requirements to be validated. we found that following tests were applicable and adequate:

   3.1. Manual code reviews and automated code inspections

   3.2. Code analysis for general programming vulnerabilities

   3.3. Component dependency analysis

   3.4. Owasp evaluation for common web vulnerabilities

   3.5. Manual pentesting

To summarize the list of intended tests during our Secure SDLC:

1. SDLC general principles and assumptions
2. Security acceptance criteria for project requirements
3. Development environment description
4. Architecture review
5. Business requirements review
6. Security requirements review
   a. Threat model
   b. Abuse cases
   c. Code analysis
   d. Component dependency analysis
   e. OWASP evaluation for common vulnerabilities (Top 10)
7. Pentesting on a particular set of features

Findings

Test results are described in this security report and are considered a project deliverable, along with the functional testing results.

Findings summary

Vulnerabilities

Severity	Critical	High	Medium	Low	Informational
# identified	1	2	1	4	1
# mitigated	-	-	-	1	-

Bugs

Severity	Critical	High	Medium	Low	Informational
# identified	1	0	0	4	1
# mitigated	-	-	-	3	1

Code smells

These affect the overall maintainability of the system.

Severity	Blocker	High	Medium	Low	Informational
# identified	0	5	11	2	0
# mitigated	-	1	-	-	-

Dependencies

Severity	Critical	High	Medium	Low	Informational
# identified	0	1	0	2	0
# mitigated	-	-	-	1	-

High level recommendations

Although the application was not tested in a production environment, it is clear that additional configurations and setup are still required for a secure setup. Taking into consideration our findings, we would advise (ourselves) to:

- Improve framework security configurations.
  - Although Django is a powerful and secure tool, it still requires proper setup
  - Django security concerns are well documented and should be followed
- Harden the development environment to disable relaxed coding standards and practices
- In addition to static code testing, linting and code reviewing, incorporate unit and integration testing into the development process, ensure auxiliary tools are useful for early detection of software flaws.
  - Ensure adequate level of test code coverage
- Implement secure headers, namely CSP Headers and actively monitor logs and CSP reports for potential exploit attempts.
- Maintain regular dependencies checks. Address all dependencies for which vulnerabilities have been found and explore potential replacements or more secure versions.
- Recommendations for future production environment:
  - Deploy Web Application Firewall solution to detect any attempts at malicious manipulations.
  - Enforce HTTPs on all communications from clients and the blog application
  - Use a scalable, secure DBMS like PostgreSQL in staging and production environments, as this is a more suitable solution for security concerned environments
  - Establish secure connections when communicating with the database container.

Finding details

Replay Attack on Comment submissions

Severity: Critical

Categories: Bug, Vulnerability

Location: Any blog post that currently accepting comments

Description: A replay attack is a form of network attack in which valid data transmission is maliciously or fraudulently repeated or delayed. Typically carried out either by the originator, or by an adversary who intercepts the data and re-transmits it, possibly as part of a spoofing attack.

The vulnerability derives from an implementation bug in the *view* responsible for handling new comment submissions. After successful submission, the server does not perform an internal redirection and, instead, serves new content in-place. This causes the browser to perform a resubmission on page refresh (POST on reload), which can lead users to post multiple comments unwillingly or an adversary to be able to spam comments, despite the measures already in place.

Recommendation: Perform code review to identify the issue and reimplement the portion of the code responsible for form processing and associated content moderation. The application should redirect the browser to *itself*, so that the browser refresh requests are not piggybacking on the previous operation. The following guide provides a detailed explanation on the solution: Prevent POST on reload.

Improper Neutralization of Special Elements

Severity: High

Categories: Vulnerability, Dependency

Location: Framework component

Description: A software flaw on an input escaping library/component was detected, which may lead to XSS vulnerabilities, if exploited. These were detected by code and dependency analysis and related to CWE-79: Improper Neutralization of Input During Web Page Generation and CWE-80: Improper Neutralization of Script-Related HTML Tags in a Web Page.

Proof: Provided in relevant outputs - Bandit.

Recommendations: Perform code review to identify the issue and fix the software vulnerability. If fixing is possible, due to third party restrictions, recommendation is to upgrade to a safer library or a newer version of the component

---

Missing security headers

Severity: High

Categories: Vulnerability

Description: HTTP security headers provide an additional layer of security by restricting behaviors that the browser and server allow once the web application is running. Implementing appropriate security headers is a best-practice for web applications and a crucial mechanism to prevent or mitigate potential attacks. This is considered of High importance due to the security goals of this application.

Recommendations: Implement X-XSS-Protection, X-Content-Type-Options, X-Frame-Options, Content-Security-Policy, Access-Control and Cross-Origin headers.

Cross-Domain Misconfiguration

Severity: Medium

Categories: Vulnerability

Description: A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.

Recommendation: Ensure that the HttpOnly flag is set for all cookies.

Potential DOM-XSS in administration pages

Severity: Low

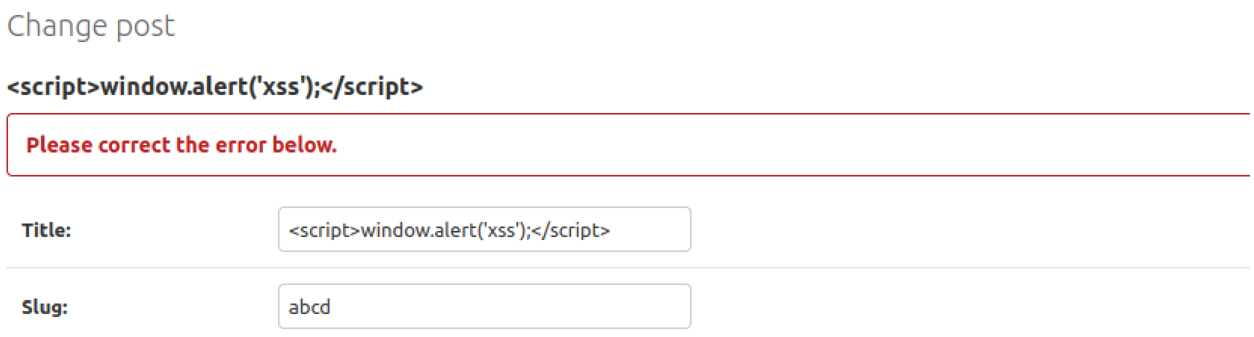
Categories: Vulnerability

Location: /admin/blog/post/

Description: DOM Based XSS is a form of XSS where the entire tainted data flow from source to sink takes place in the browser, i.e., the source of the data is in the DOM, the sink is also in the DOM, and the data flow never leaves the browser. These types of attack can rewrite content of the HTML page and adversaries may leverage these vulnerabilities to send malicious scripts to users.

The vulnerability is listed as potential with a low severity because the browser did not execute the payload, although the page was slightly **defaced**, including a form field, page title and browser tab. The page also became slower for larger payloads.

Proof:


Change post

<script>window.alert('xss');</script>

Please correct the error below.

Title:	<script>window.alert('xss');</script>
Slug:	abcd

Recommendations:

Generally, preventing XSS vulnerabilities is the combined result of a set of different measures, such as:

- Input sanitization and filtering on arrival. When received by the server, filter and validate data
- Output encoding. Ensure server provided content is properly encoded to plain, text or html encodings, preventing unintentional script execution by the browser
- Use appropriate response headers and content policies. Recommendation is to implement and actively maintain CSP policies to reduce the severity of the attack, in the likelihood of an occurrence.

Erroneous username assignment on blog comments

Severity: Low

Categories: Bug

Status: Fixed

Description: Comment submission is improperly handled in the backend and the user is always displayed as the default user, Anonymous, in the blog application.

Recommendations: Fix the corresponding *view* and *form* controllers to ensure user assignment is correctly performed on form submission.

Server-side information disclosure via response Headers

Severity: Low

Categories: Vulnerability

Description: The web application is leaking version information via the server HTTP response header. Access to this information may facilitate adversaries identifying other vulnerabilities the web application or server are subject to.

Recommendation: Suppress the “Server” HTTP Header or provide only generic information.

Strict-Transport-Security Header not set

Severity: Low

Categories: Vulnerability

Description: HTTP Strict Transport Security (HSTS) is a web security policy mechanism whereby a web server declares that complying user agents (such as a web browser) are to interact with it using only secure HTTPS connections (i.e. HTTP layered over TLS/SSL)

Recommendation: Ensure the server, application server, load balancer, etc. is configured to enforce Strict-Transport-Security.

Cookie No HttpOnly Flag

Severity: Low

Categories: Vulnerability

Description: A cookie has been set without the HttpOnly flag, which means that the cookie can be accessed by JavaScript. If a malicious script can be run on this page then the cookie will be accessible and can be transmitted to another site. If this is a session cookie then session hijacking may be possible.

Recommendation: Ensure that the HttpOnly flag is set for all cookies.

Improper XML / HTML definitions

Severity: Low

Status: Fixed

Categories: Bug

Description: Code review tools identified a set of major bugs related to potentially erroneous html and xml definitions. These were not considered major application flaws and are categorized here as low impact.

The followings bugs were identified by SonarCloud, mostly related to the

- Add "lang" and/or "xml:lang" attributes to this "<html>" element
    - *templates/base.html*
    - *templates/posts.html*
- Insert a <!DOCTYPE> declaration to before this <html> tag"
    - *templates/posts.html*
- Add a <title> tag to this page
    - *templates/posts.html*

Recommendation: Most of these pertain to unused template files that remained in the codebase during the development cycle. *Posts.html* has been removed. In any case, suggestion is to fix all recommendations and ensure templates are correctly created, following best practices for HTML documents.

Cross-Domain Javascript File Inclusion

Severity: Informational

Categories: Vulnerability, Dependency

Description: Pages include one or more script files from third-party domains. This is due to the test environment, which was a development environment and not all web dependencies were being statically served from the app's server trusted source.

Recommendation: Ensure Javascript source files are loaded only from trusted sources that cannot be controlled by application end users. Serve static files when in production or staging.

Assessment of function and security requirements

The focus of a threat and countermeasure categorization is to define security requirements in terms of the threats and the root cause of the vulnerability. A threat can be categorized by using STRIDE, an acronym for **Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, and Elevation of privilege**.

The root cause can be categorized as

- security flaw in design,
- a security bug in coding, or
- an issue due to insecure configuration.

Each threat will then lead to one of these 3 phases of the SDLC.

Abuse Cases

The table below summarizes the requirements in scope for the Abuse case analysis.

ID	Name	Requirement
RF.1 – A	Content management service (CMS)	Service to manage own blog posts and comments
RF.1 – B	Blog Post Management	Allow users to create, edit and delete their own blog posts
RF.1 – B.1	Create Blog Post	Input screens for creating blog posts.
RF.1 – B.2	Edit Blog Post	Input screens for editing existing blog posts
RF.1 – B.2.1	Manage Blog Post visibility	Allow users to change the blog post status between the following states: 1. Draft – Only visible to the author 2. Published – Publicly visible
RF.1 – B.3	Delete Blog Post	Input screens for removing blog posts
RF.1 – C	Comments on Blog Posts	Support both the adding and moderation of comments on blog posts
RF.1 – C.1	Add comments	Allow site visitors and authenticated users to comment on blog posts.

RF.1 – C.2	Content moderation	Moderate comments before they are publicly displayed
RF.1 – C.2.1	Comment visibility	Approve or reject comments to their blog posts.
RF.1 – C.2.2	Add comments	Input screens for adding new comments. Authenticated and anonymous site visitors shall be able to comment on blog posts.
RF.1 – C.2.3	Remove comments	Authors have the ability to remove comments on their blog posts.

The table below documents the identified abuse cases and the respective feature, in accordance with the previous requirements table and OWASP recommendations.

Abuse Case ID	**Feature ID**	**Abuse Case Description**	**Countermeasure**
ABUSE_CASE_001	RF.1 - A	Gain unauthorized access to another user's CMS	Password based authentication mechanisms
ABUSE_CASE_002	RF.1 - B	The web server could be subject to a cross-site scripting attack because it does not sanitize untrusted input.	Input validation and sanitization
ABUSE_CASE_003	RF.1 - A RF.1 - B RF.1 - C	Adversaries may attempt Client-Side Injection attacks to introduce malicious content in the database	Use Django's built-in ORM. Access database only using QuerySets and secure mechanisms provided by the development framework.
ABUSE_CASE_004	RF.1 - B.2	By editing the files directly with a hex editor we can bypass the permissions for the object	Run your code using the lowest privileges that are required to accomplish the necessary tasks; Separation of Privilege: Identify the functionality that requires additional privileges, such as write access to configuration files, and isolate the process that has write privileges.

			Raise privileges as late as possible, and drop them as soon as possible. Run integrity checking on File System critical files in order to avoid disrupting by undesired writings.
ABUSE_CASE_005	RF.1 - B.2.1	Browser Client may be able to impersonate the context of Human User in order to gain additional privilege.	Avoid impersonation configurations on client side and on web servers. Force users authenticated and authorized access on all layers of the services to enable any incorrect privilege.
ABUSE_CASE_006	RF.1 - C.2.1 RF.1 - C.2.2 RF.1 - C.2.3	If Add/Remove Comment Post is given access to memory then Add/Remove Comment Post can be tampered	
ABUSE_CASE_007	RF.1 - B.3	File System may be spoofed by an attacker and this may lead to data being written to the attacker's target	Least privilege policy on processes and on data access within the app file system. Define protection ‘hard limits’ for storage usage for all users of the system, log files, etc..., avoiding disk exhaustion.

ABUSE_CASE_008	RF.1 – B	Improper data protection of File System can allow an attacker to read information not intended for disclosure	authorization mechanisms on all access to data, without exception or elevation of privileges on system file access.
ABUSE_CASE_009	RF.1 – A RF.1 – B	Cookies may be spoofed by an attacker and this may lead to data being written to the attacker's target instead of Cookies	cookies should have encrypted data, with nonce or similar mechanisms to avoid reuse of past cookies or other users cookies. Sessions are resistant to replay attacks.
ABUSE_CASE_010	RF.1 – A RF.1 – B	Cookies may be spoofed by an attacker and this may lead to incorrect data delivered to Browser Client	same as ABUSE_CASE_009
ABUSE_CASE_011	RF.1 – A	Improper data protection of Cookies can allow an attacker to read information not intended for disclosure.	Cookies should have encrypted data (No sensitive information is stored in clear text in the cookie) Protect secrets, don’t use obfuscation tactics with cookies.The contents of the authentication cookies are encrypted.
ABUSE_CASE_012	RF.1 – C.1	An attacker may pass data into Browser Client in order to change the flow of program execution	Architecture should not allow direct use of paths, be it relative or absolute, in file system or in http request. ainda estou a pensar, pesquisar e escrever

		within Browser Client to the attacker's choosin	
ABUSE_CASE_013	RF.1 - B.1 RF.1 - B.2	Blog App may be able to impersonate the context of Create/Edit Post in order to gain additional privilege.	Same as ABUSE_CASE_005.
ABUSE_CASE_014	RF.1 - C.1	Packets or messages without sequence numbers or timestamps can be captured and replayed in a wide variety of ways.	Use communication protocols and APIs that have protection mechanisms against replay of messages.

Threat Modeling

Threat modeling is a popular technique to help system designers and software architects to think about the security threats that their systems and applications might face. Threat modeling can be seen as risk assessment for applications and it aids the design and development of appropriate mitigation strategies for potential vulnerabilities.

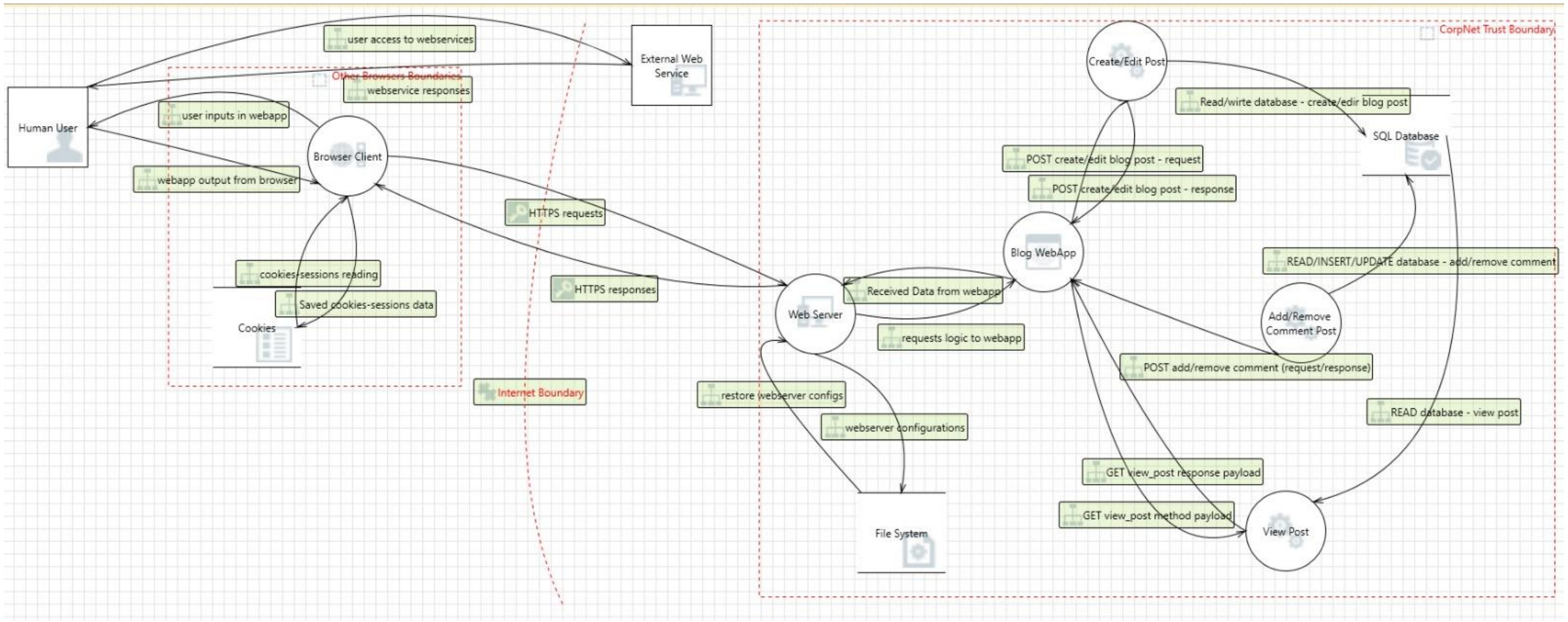
Upon collecting the required information for performing threat modeling, including software specification, use cases, system agents, dependencies, abuse cases, etc, we developed a set of Data Flow Diagrams, modeling the flow of information and interactions within the system.

Using Microsoft's Threat Modeling Tool we modeled the most relevant interactions with both web applications, the blog itself and the content management system. This process helped us identify critical components and their exposure to threats.

This section details our threat modeling efforts and our analysis on the major threats our application may be exposed to.

Blog web application

The diagram below displays both the server and client applications modeled together


user access to webservices
External Web Service
Other Browsers Boundaries
webservice responses
Human User
user inputs in webapp
Browser Client
webapp output from browser
HTTPS requests
cookies-sessions reading
Saved cookies-sessions data
Cookies
HTTPS responses
Internet Boundary
CorpNet Trust Boundary
Create/Edit Post
Read/wirte database - create/edir blog post
SQL Database
POST create/edit blog post - request
POST create/edit blog post - response
Blog WebApp
READ/INSERT/UPDATE database - add/remove comment
Received Data from webapp
Web Server
requests logic to webapp
Add/Remove Comment Post
POST add/remove comment (request/response)
restore webserver configs
webserver configurations
READ database - view post
GET view_post response payload
GET view_post method payload
View Post
File System

Server Side in more detail:

On the server side, the main vulnerability points were included in our model, and more lateral or less relevant dependencies were not considered at all (like for example AAA mechanism in Server SO, that is assumed to be out-of-scope).

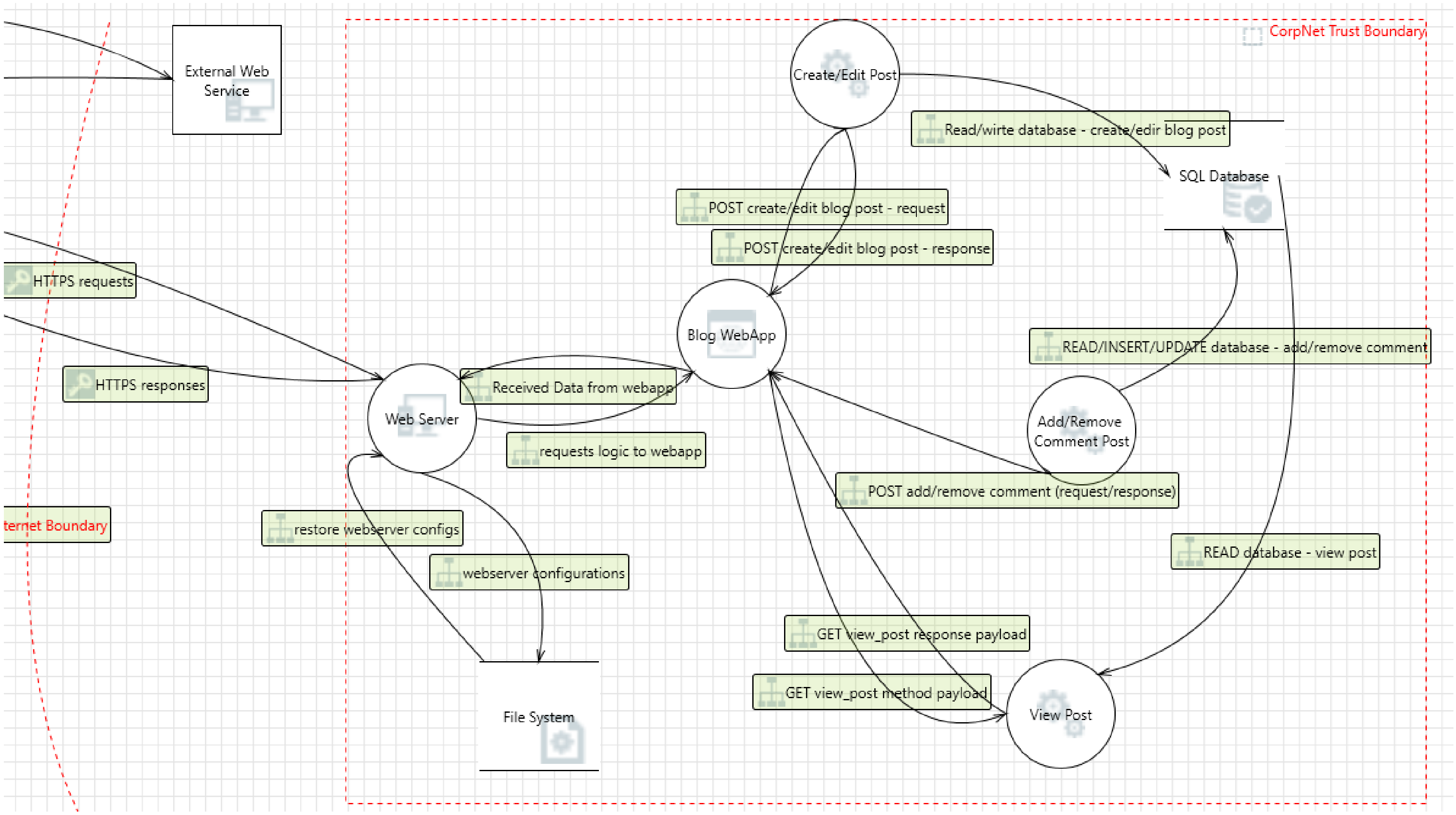
The main components included in technological terms are:

- Web server,
- Server's filesystem
- The *webapp* methods. viewed separately because they can imply different implementations and can have different levels of risk for the system.
- Web services infrastructure and external web services (located outside of the trust boundary of the server side of our application - considered out-of-scope for analysis, but included for modeling purposes)
- Database instance used for this app

Data flows were detailed and named according to their semantics and also to the direction between components.

In terms of boundaries, a unique trust boundary was considered for the all solution on server side, except for the web services support of the app generated by the framework and that is supposed to be installed in dmz-like zone, less trustable but not very relevant because it was also not used on our proof-of-concept app.

For each component, security properties and other detailed definitions on each entity or data flow has been reflected in the tool, to be able to have a more accurate threat list at the end.


External Web Service
CorpNet Trust Boundary
Create/Edit Post
Read/wirte database - create/edir blog post
SQL Database
POST create/edit blog post - request
POST create/edit blog post - response
HTTPS requests
Blog WebApp
READ/INSERT/UPDATE database - add/remove comment
HTTPS responses
Received Data from webapp
Web Server
Add/Remove Comment Post
requests logic to webapp
POST add/remove comment (request/response)
ternet Boundary
restore webserver configs
webserver configurations
READ database - view post
GET view_post response payload
GET view_post method payload
View Post
File System

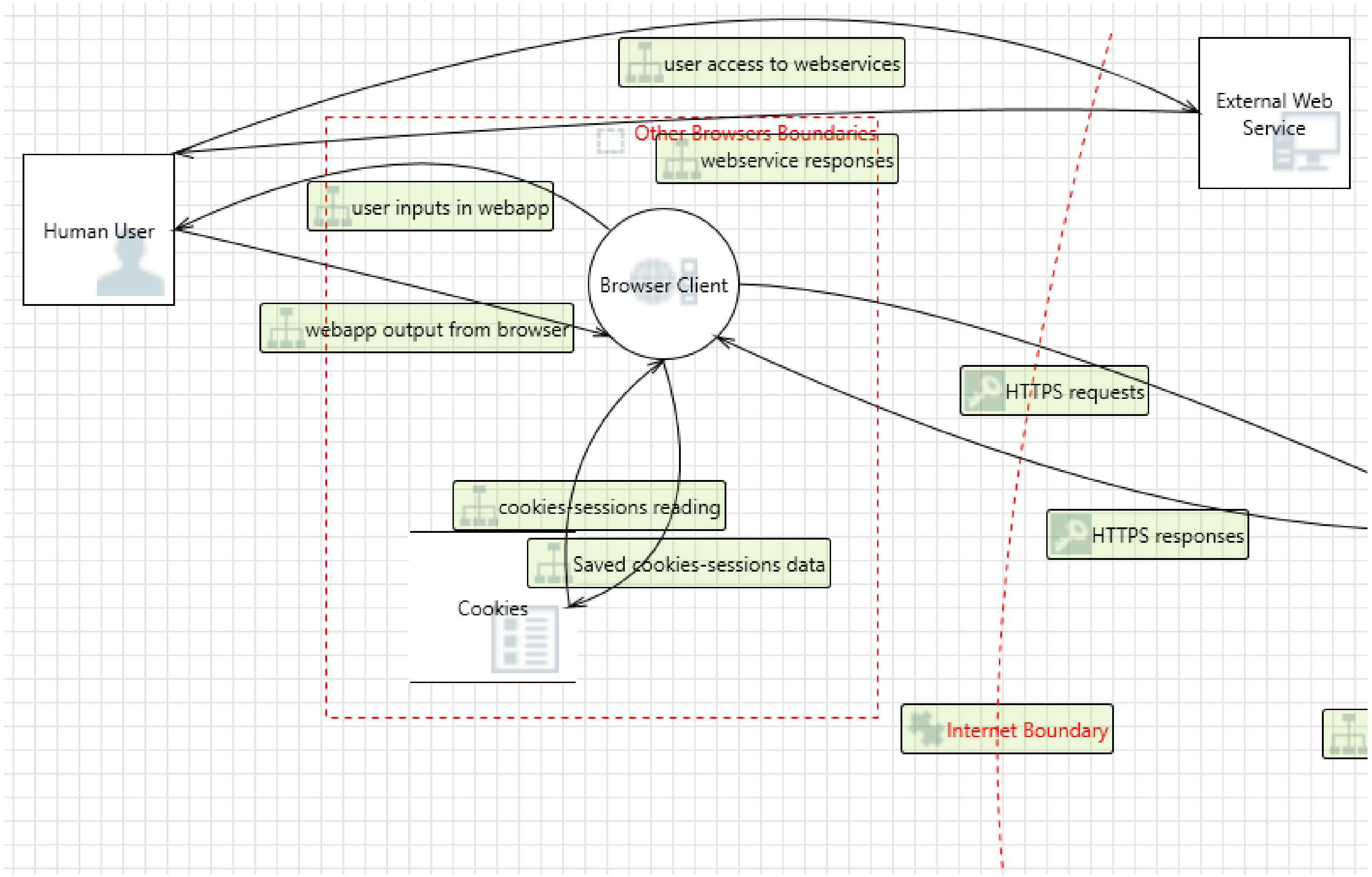
Client side

On the client side, the same approach was followed - choosing relevant components of the solution in terms of security.

The main components included in technological terms are:

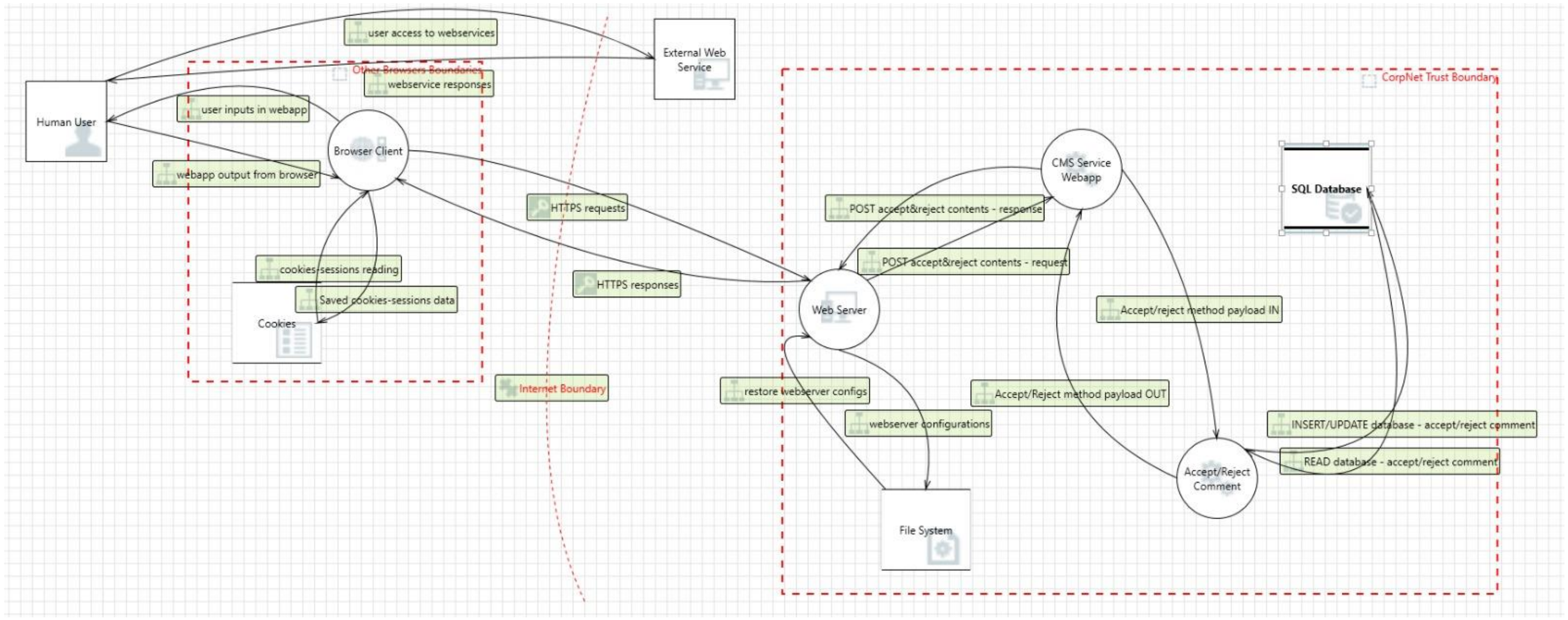
- Browser instance
- Cookies (considered as a point of persistence)
- The human user

An internet boundary between client and server was defined, which gives the tool the information for all the normal risks associated with this type of boundary.


user access to webservices
External Web Service
Other Browsers Boundaries
webservice responses
Human User
user inputs in webapp
Browser Client
webapp output from browser
HTTPS requests
cookies-sessions reading
HTTPS responses
Saved cookies-sessions data
Cookies
Internet Boundary

Content Management System

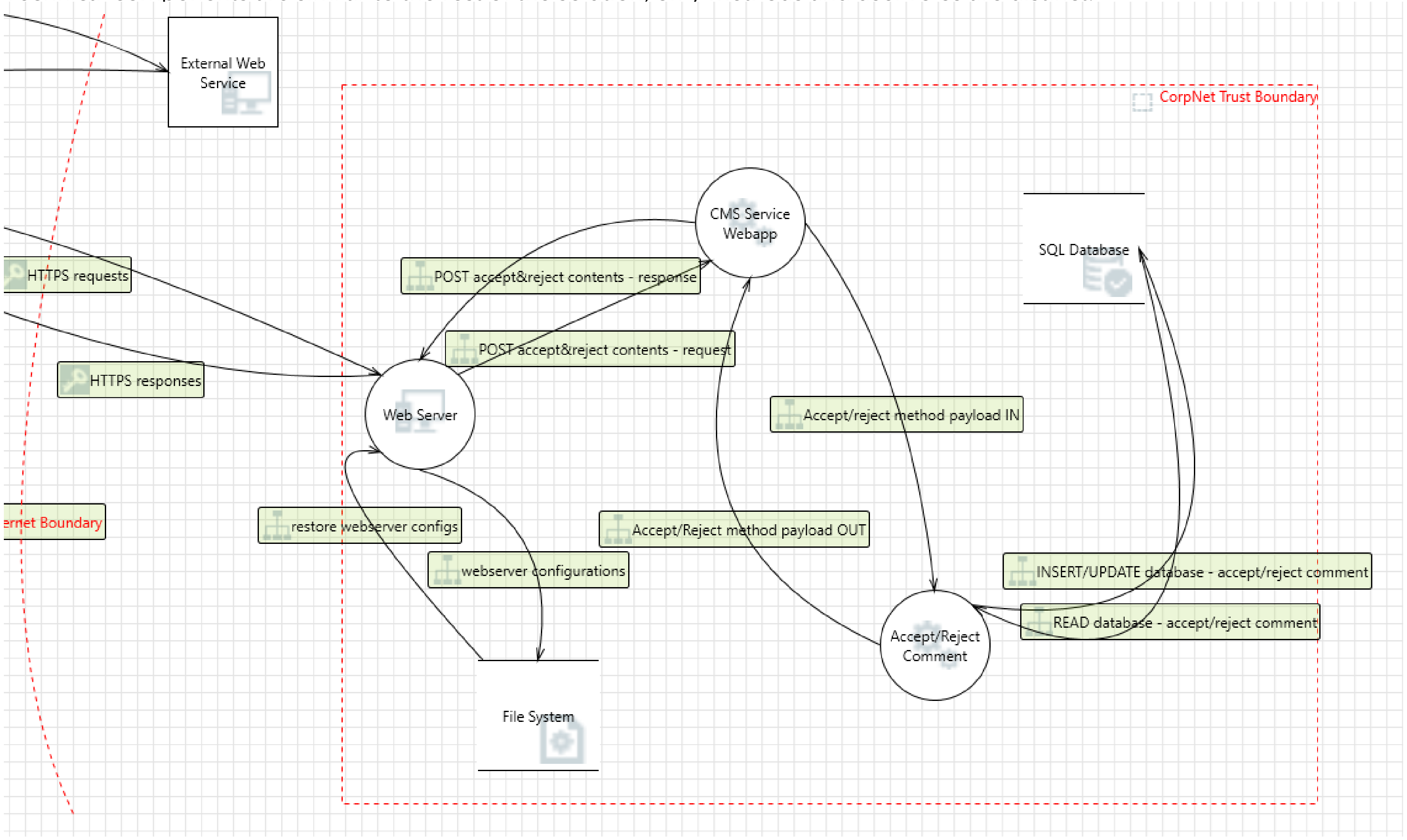
Diagram modeled with both client and server side.


user access to webservices
External Web Service
Other Browsers Boundaries
webservice responses
Human User
user inputs in webapp
Browser Client
webapp output from browser
HTTPS requests
cookies-sessions reading
Saved cookies-sessions data
Cookies
HTTPS responses
Internet Boundary
CorpNet Trust Boundary
CMS Service Webapp
SQL Database
POST accept&reject contents - response
POST accept&reject contents - request
Accept/reject method payload IN
Web Server
restore webserver configs
webserver configurations
Accept/Reject method payload OUT
INSERT/UPDATE database - accept/reject comment
READ database - accept/reject comment
Accept/Reject Comment
File System

In more detail, on the server side of CMS app (client side is the same as for BlogApp client):

The approach for this 2nd server side diagram is in fact because it is focused on the CMS app, which is a separated front-end, existing in the server side for administration purposes.

Technical components are similar to the rest of the solution, only methods and user roles are distinct.


External Web Service
CorpNet Trust Boundary
CMS Service Webapp
SQL Database
HTTPS requests
POST accept&reject contents - response
POST accept&reject contents - request
HTTPS responses
Web Server
Accept/reject method payload IN
ernet Boundary
restore webserver configs
Accept/Reject method payload OUT
webserver configurations
INSERT/UPDATE database - accept/reject comment
READ database - accept/reject comment
Accept/Reject Comment
File System

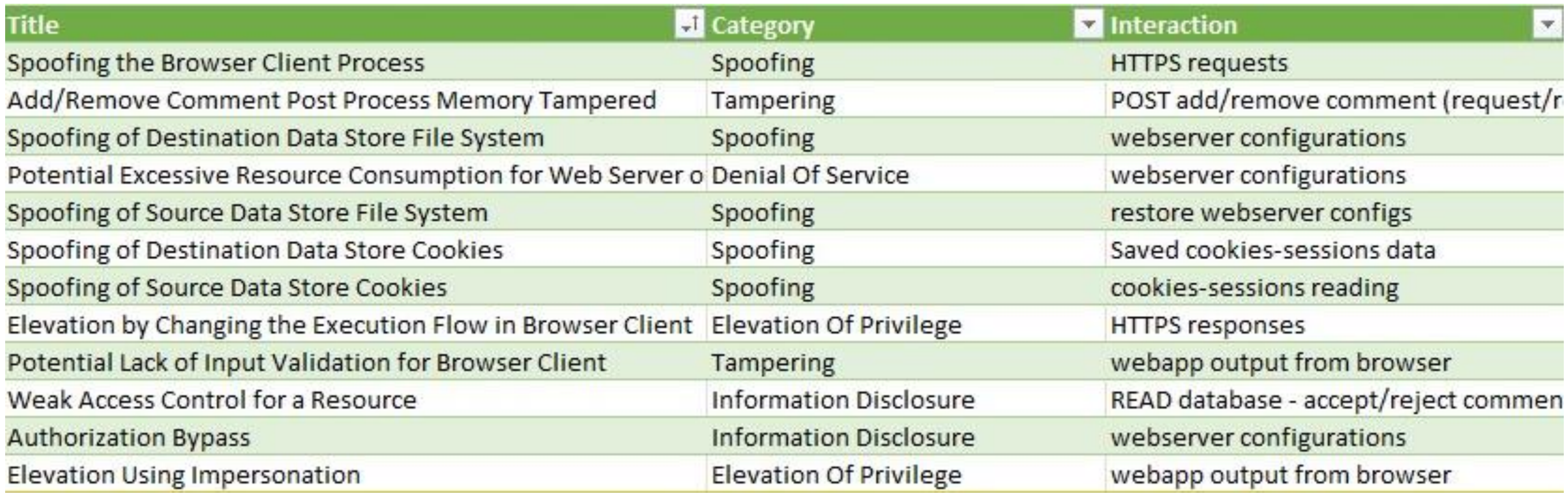
After a risk analysis of the list of threats generated by the tool, some of the threats identified have been classified as LOW or MEDIUM RISK, and excluded from our abuse cases and security requirements or testing scope.

High risk threats

After iterating over the threat model and adjusting both requirements and architecture details, we prioritized the final risk list.

This task had 2 aspects:

- Eliminating 'duplicate' threats (methods that have similar risk level and same implementation context)
- evaluation risk for each threat – excluding threats that are less probable or have low impact on security requirements



Title	Category	Interaction
Spoofing the Browser Client Process	Spoofing	HTTPS requests
Add/Remove Comment Post Process Memory Tampered	Tampering	POST add/remove comment (request/r
Spoofing of Destination Data Store File System	Spoofing	webserver configurations
Potential Excessive Resource Consumption for Web Server o	Denial Of Service	webserver configurations
Spoofing of Source Data Store File System	Spoofing	restore webserver configs
Spoofing of Destination Data Store Cookies	Spoofing	Saved cookies-sessions data
Spoofing of Source Data Store Cookies	Spoofing	cookies-sessions reading
Elevation by Changing the Execution Flow in Browser Client	Elevation Of Privilege	HTTPS responses
Potential Lack of Input Validation for Browser Client	Tampering	webapp output from browser
Weak Access Control for a Resource	Information Disclosure	READ database - accept/reject commen
Authorization Bypass	Information Disclosure	webserver configurations
Elevation Using Impersonation	Elevation Of Privilege	webapp output from browser

Security requirements

Based on initial requirements defined in the early stage of the design, and after the threat model analysis and abuse cases identification, we summarize a set of additional security requirements.

The table below represents an assessment of the security requirements, detailed under **RF.2 – Security Requirements** in the Architecture documentation.

ID	Assessment	Risk Level	Requirement
A	Not implemented at server level between components (e.g. web application to database)	High risk	Secure communication
A.1.	We feel HTTPS is very important to ensure safe communication and protect user data while in transit. Without it, the system may become susceptible to adversaries trying to eavesdrop or tamper with the communication between a client and the application server. This is recommended for staging and production envs.	High risk	HTTPS
B	Without secure data storage, data breaches or unauthorized access to information can occur.	High risk	Secure data storage
C	Bad content security poses a relative risk of unauthorized access and content manipulation	Medium risk	Content security
C.1.	Poses a high risk of injection attacks and unauthorized access.	High risk	Input validation
C.2.	There is a medium risk of spam content infiltrating the system and affecting the performance of the system	Medium risk	Anti-spam
C.3.	High risk of script execution. It can cause theft of data, session hijacking and unauthorized access to user accounts.	High risk	XSS injection
D	There is some risk that outdated libraries can be exploited by	Medium risk	Dependencies

	attackers		
E	If we don’t use proper authentication it can lead to unauthorized access and data leaks	High risk	Secure authentication
F	Bad logging and error handling can make incident detection and response a complicated task	Medium risk	Logging and Error handling
G	We need to update our system regularly so it doesn’t lead to vulnerabilities being exploited	High risk	Security Updates and Patch Management
H	High risk of unauthorized access or data leaks	High risk	Credential Management
H.1.	Implementation of password policies on user registration helps protect user accounts from brute force attacks on user credentials	Medium risk	Strong Password Policy
H.2.	Storing plain texts of passwords can grant a high risk of unauthorized access to user passwords	High risk	Secure Password storing

Appendices

Severity and risk scoring

Critical – Poses an immediate threat to key processes or resources.

High – Poses a direct threat to the application or resources.

Medium – Findings who may lead to indirect threats to key processes.

Low – There is not a direct threat associated with the finding. Vulnerabilities may be exploited in combination with other vulnerabilities.

Informational – This type of finding does not represent a vulnerability, but states an issue on security principles, a design flaw or an improper implementation that may cause problems.

Assessment metrics

The Common Vulnerability Scoring System (CVSS version 3.1) is adopted to calculate vulnerability impact. For the purpose of this testing effort, we shall only consider the following metrics:

- Attack Vector (AV)
  - Network (N): We shall test and consider potentially vulnerable components bound to the network stack and the set of possible attackers extends beyond the other options listed below, up to and including the entire Internet.
- Attack complexity (AC)
  - All metrics: We shall consider Low (L) and High (L), as both are relevant to the project's scope. Low difficulty attacks are especially relevant, as these may allow repeated successful attacks, should an adversary find a vulnerable component.
- Privileges Required (PR):
  - Low (L) and None (N): We are mainly concerned about low privileged attacks.
- User interaction (UI):
  - None (N): We shall consider attacks that do not require prior user interaction.
  - Required (R): We shall consider attacks that require user interaction at administrator level. For instance, attacks arising from misconfigured user

permissions by a system administrator or misconfigurations during an installation process. We shall not consider attacks that require an adversary to engage with regular application users.

- Impact Metrics:
    - Confidentiality (C): We shall consider any breach of confidentiality as relevant for the scope of this analysis.
    - Integrity (I): We shall consider any breach of confidentiality as relevant for the scope of this analysis.
    - Availability (A): (High) We shall only consider attacks with high impact on system availability.

External tools used and relevant outputs

Bandit

Bandit is a security linter for python that aims to find common security issues in Python code.

Bandit was used to perform regular security checks on SecureBlog's code, ensuring no major flaws were detected.

Bandit was able to detect potential medium and high impact vulnerabilities in the code, displayed in the table below:

Issue	Description	CWE	Severity	Confidence
B703:django_mark_safe	Potential XSS on mark_safe function	CWE-80	Medium	High
[B308:blacklist]	Use of mark_safe() may expose cross-site scripting vulnerabilities and should be reviewed	CWE-79	Medium	High

Both vulnerabilities relate to a template tag for displaying markdown content. The initial implementation escaped content as follows:

mark_safe(markdownify(text))

Safety indicates this may lead to potential XSS attacks, as the usage of the **marksafe** method is blacklist in the tool's configurations.

Safety

Safety checks Python dependencies for known security vulnerabilities and suggests the proper remediations for vulnerabilities detected. It was used to assess the project requirements' file: *requirements.txt*. The tool never did pick up any severe vulnerabilities and after some initial changes to dependencies it no longer identified any vulnerable dependencies in the requirements.txt. Although slightly odd, the fact we intentionally limited the number of external dependencies used may also contribute to these results.

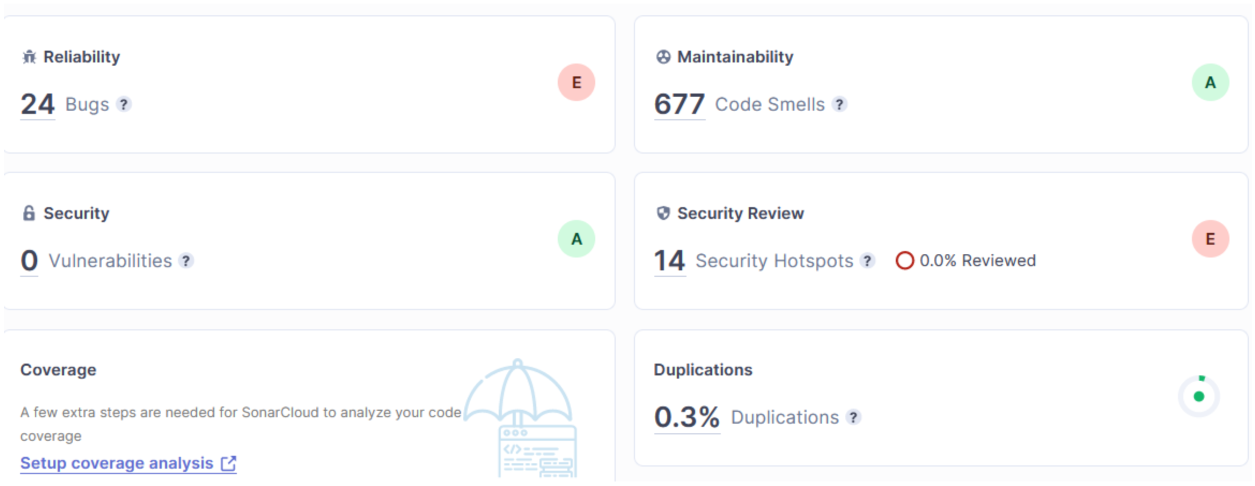
The latest scans from safety are made available annex to this report, under *outputs/safety.json.*

SonarCloud

SonarCloud is a renowned tool for static code analysis, capable of identifying bugs, vulnerabilities, code smells and security hotspots in software projects. SonarCloud was integrated with the project's repository in order to perform regular automated code reviews.

Security reports obtained were always quite positive, this is also likely due to the limited number of features implemented, but a good result nonetheless. As the project's source was always submitted to *Github* without static assets, many of the identified issues relate directly to the implementation *javascript* and *css* libraries and, as specified in Limitations, these will not be in scope for this analysis.

Initial analysis provided results similar to the one provided below. These analyses took into account the entirety of the source, resulting in upwards of 13k lines of code.


Reliability
24 Bugs ? E

Maintainability
677 Code Smells ? A

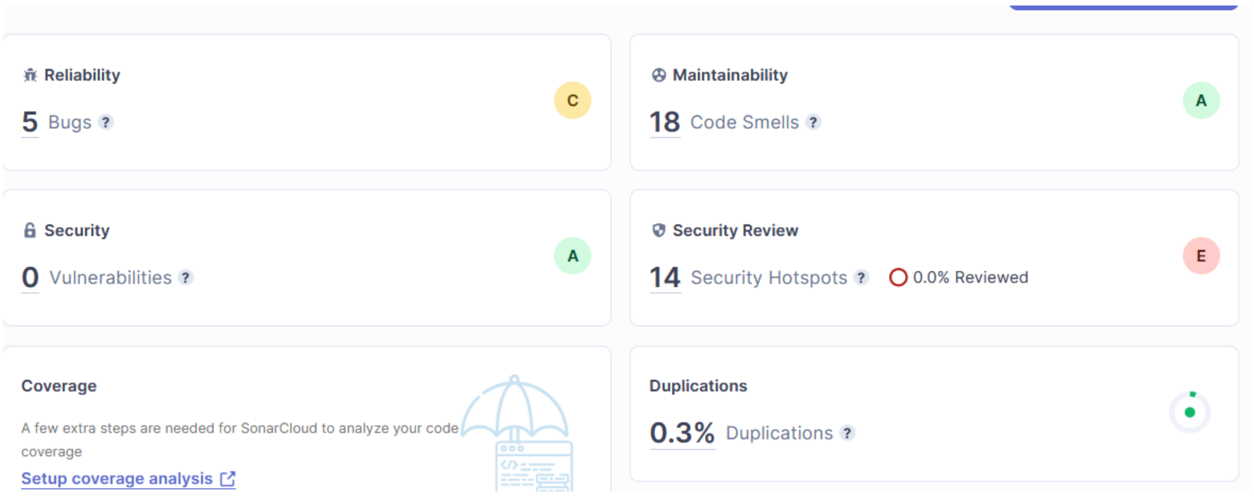
Security
0 Vulnerabilities ? A

Security Review
14 Security Hotspots ? 0.0% Reviewed E

Coverage
A few extra steps are needed for SonarCloud to analyze your code coverage
Setup coverage analysis

Duplications
0.3% Duplications ?

Subsequent analysis, after additional configuration and file exclusions, limited the scope to only a subset of the initial results, only pertaining to python and html files.


Reliability

5 Bugs ?

C

Maintainability

18 Code Smells ?

A

Security

0 Vulnerabilities ?

A

Security Review

14 Security Hotspots ? 0.0% Reviewed

E

Coverage

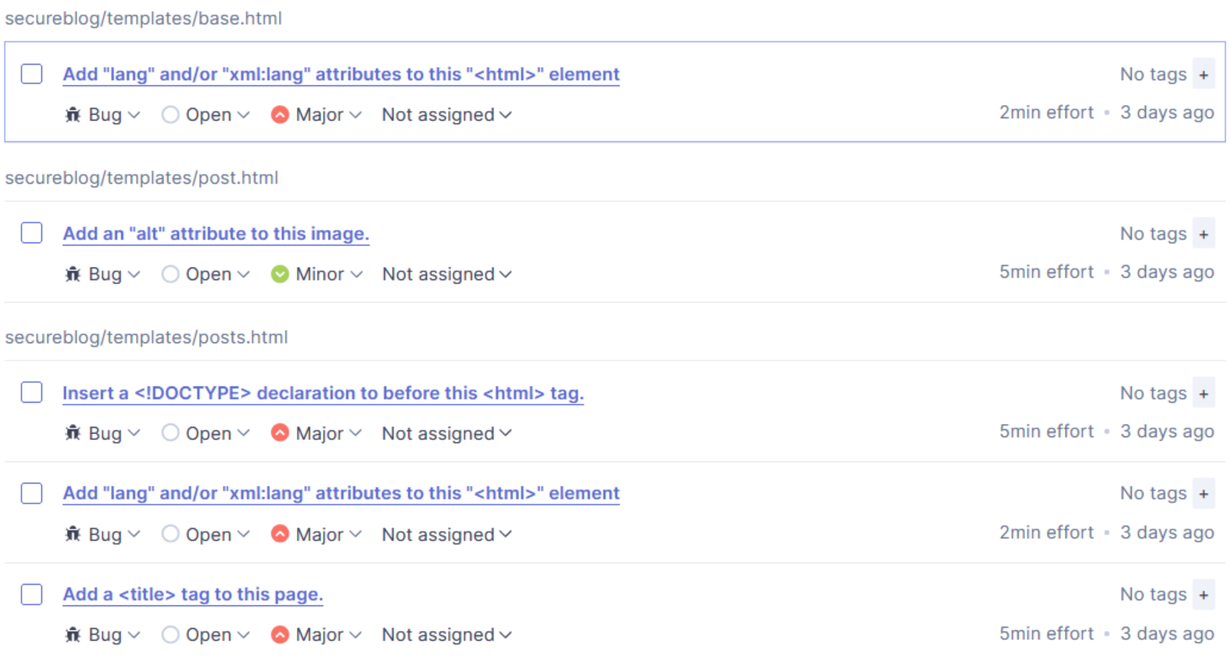
A few extra steps are needed for SonarCloud to analyze your code coverage

Setup coverage analysis

Duplications

0.3% Duplications ?

The following bugs were identified, major flaws are documented under *Finding Details.*


securebloq/templates/base.html

Add "lang" and/or "xml:lang" attributes to this "<html>" element	Bug · Open · Major · Not assigned	No tags · 2min effort · 3 days ago

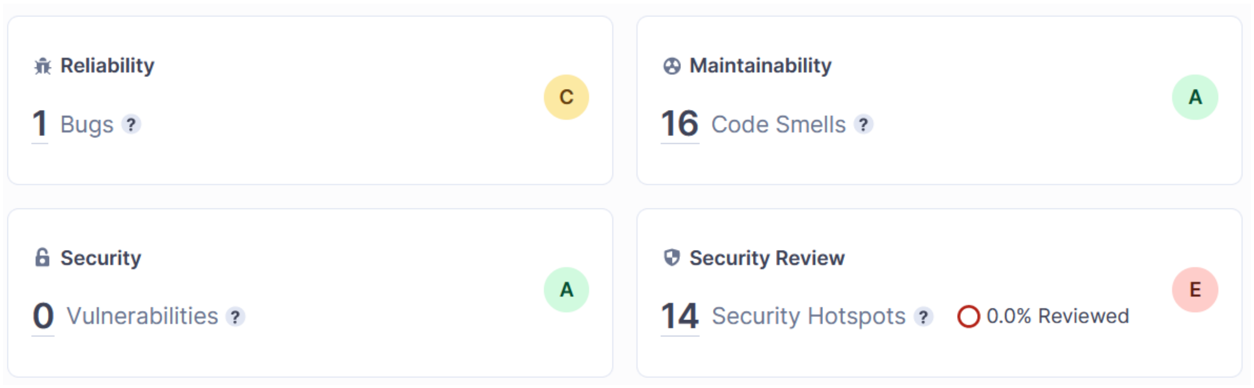
securebloq/templates/post.html

Add an "alt" attribute to this image.	Bug · Open · Minor · Not assigned	No tags · 5min effort · 3 days ago

securebloq/templates/posts.html

Insert a <!DOCTYPE> declaration to before this <html> tag.	Bug · Open · Major · Not assigned	No tags · 5min effort · 3 days ago
Add "lang" and/or "xml:lang" attributes to this "<html>" element	Bug · Open · Major · Not assigned	No tags · 2min effort · 3 days ago
Add a <title> tag to this page.	Bug · Open · Major · Not assigned	No tags · 5min effort · 3 days ago

These mostly relate to HTML formal specification and proper XML declaration and were not deemed essential for the security related objectives of this project. These are still detailed under Finding Details, as low impact bugs. These also pertain to two unused templates, from previous implementation cycles. After removing the lingering templates, there is only 1 bug remaining, in the *base.html* file.



In the case of **code-smells** SonarCloud identifies 4 critical issues and 12 major issues, mostly pertaining to dead or unused code, which links to the CWE family of Improper Adherence to Coding Standards and CWE-1164: Irrelevant Code.

BurpSuite

BurpSuite was used for sniffing requests and attempting to exploit the application manually.. In order to proxy all traffic between our browsers to Burp we set up a Burp Proxy running on 127.0.0.1:8080 then we set Firefox proxy to our Burp proxy.

Early tests allowed us to identify two software flaws and one high impact vulnerability, detailed in Finding Details.

OWASP ZAP

OWASP ZAP is an open-source web application security scanner and one of the most active Open Web Application Security Project projects.

ZAP was used to crawl the application, detect misconfigurations and detect potential vulnerabilities via active scan. A complete ZAP report is provided under *outputs/zap_scan_report* in the submission repository. ZAP reports produced while developing the application were assessed and analyzed and some of the outputs were deemed out-of-scope for this report or given different levels of impact.

The most relevant findings are reflected in the Finding Details section of this document.