

[Do It Safe]

CASE STUDY

SECURE SOFTWARE ENGINEERING

A.A. 2020-2021

TEAM

Gasparro Paolo p.gasparro4@studenti.uniba.it

Forleo Giovanni g.forleo3@studenti.uniba.it

Shahoveisi Ardavan a.shahoveisi@studenti.uniba.it

Muppuri Karthikeyan k.muppuri@studenti.uniba.it

Index

1.	SYSTEM: CIVIC SENSE.....	4
1.1	HOW THE SYSTEM WAS.....	4
1.2	HOW THE SYSTEM IS.....	7
2.	VULNERABILITIES ANALYSIS	14
2.1	STATIC CODE ANALYSIS.....	14
2.1.1	<i>Critical vulnerabilities</i>	<i>15</i>
2.1.2	<i>High vulnerabilities</i>	<i>19</i>
2.1.3	<i>Medium vulnerabilities</i>	<i>24</i>
2.1.4	<i>Low vulnerabilities</i>	<i>26</i>
2.2	SECURITY FIX	31
2.2.1	<i>A1 Injection.....</i>	<i>31</i>
2.2.2	<i>A2 Broken Authentication</i>	<i>34</i>
2.2.3	<i>A3 Sensitive Data Exposure.....</i>	<i>35</i>
2.2.4	<i>A4 XML External Entities (XXE)</i>	<i>39</i>
2.2.5	<i>A5 Broken Access Control.....</i>	<i>39</i>
2.2.6	<i>A6 Security Misconfiguration</i>	<i>40</i>
2.2.7	<i>A7 Cross-Site Scripting (XSS)</i>	<i>41</i>
2.2.8	<i>A8 Insecure Deserialization</i>	<i>42</i>
2.2.9	<i>A9 Using Components with Known Vulnerabilities.....</i>	<i>43</i>
2.2.10	<i>A10 Insufficient Logging and Monitoring</i>	<i>43</i>
2.2.11	<i>Not Set.....</i>	<i>43</i>
2.2.12	<i>Final report</i>	<i>46</i>
2.3	VULNERABILITIES NOT IDENTIFIED BY FORTIFY.....	48
2.3.1	<i>A3 Sensitive Data Exposure.....</i>	<i>48</i>
2.3.2	<i>A5 Broken Access Control.....</i>	<i>48</i>
2.3.3	<i>A9 Using components with Known Vulnerabilities</i>	<i>51</i>
2.3.4	<i>Observations</i>	<i>52</i>
3.	PRIVACY ANALYSIS.....	53
3.1	PRIVACY ASSESSMENT	53
3.1.2	<i>Privacy Pattern applied.....</i>	<i>54</i>
3.1.4	<i>Privacy Report.....</i>	<i>58</i>
3.1.5	<i>Design Strategies</i>	<i>59</i>

3.2	EXPLAINING UNRESOLVED VULNERABILITIES WITH PRIVACY PATTERNS....	65
3.2.1	<i>Privacy Violation (Critical).....</i>	65
3.2.2	<i>JavaScript Hijacking: Vulnerable Framework (Low).....</i>	65
3.2.3	<i>Cross-Site Request Forgery (Low).....</i>	66

1. SYSTEM: CIVIC SENSE

The system allows the citizen to notify failures, problems, malfunctions and, in general, relevant events for a subject that provides services or manages infrastructure of public interest (water, electricity, roads, urban safety, etc. ...).

1.1 How The System Was

To test the system, the XAMPP tool has been used. It is an open-source software platform built on Apache HTTP Server, database MySQL and all the tools useful to use PHP programming language.

After the installation of XAMPP, the Civic Sense main folder has been placed into the 'xampp\htdocs' path. In this case, the path is 'C:\xampp\htdocs'.

Writing 'localhost/Civic-Sense' in a Web Browser Client (Google Chrome, Mozilla Firefox, Microsoft Edge, etc.), the system can be started.

The main interface of the system is shown below:

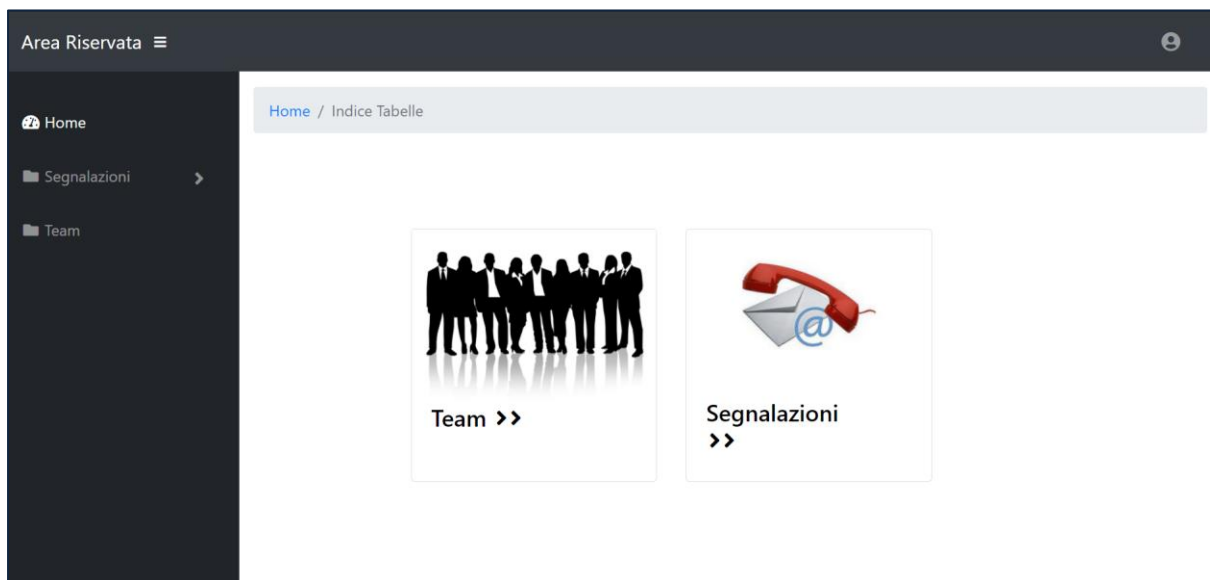


Figure 1: Civic Sense Home page

However, the system was not able to show all functionality for which it was created. Browsing through the pages, clicking the 'Team button', it was immediately clear that a database with some data was missing, as shown below:



Figure 2: Team main page (database connection error)

Clicking on 'Segnalazioni' in the main page, the system did not seem to work properly.



Figure 3: Segnalazioni main page (no content shown)

Clicking, instead, on ‘Segnalazioni’ button on the left part of the page, the dropdown menu shows the five sub-categories:

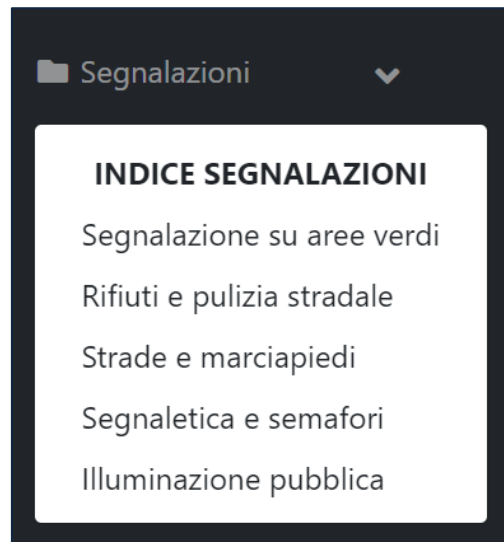


Figure 4: Segnalazioni's five sub-categories

However, clicking on each sub-category, it was evident that also in this case the table was missing:



Figure 5: Example of empty table of a sub-category

1.2 How The System Is

It was evident that the system had to be fixed in some ways to let the team understand the full functionalities of it and act consequently (even with vulnerability fixes).

To work on group and track all the modifications remotely, it has been used the GitHub as versioning tool. The private repository is placed to this URL: <https://github.com/pegasusgio/Civic-Sense/>.

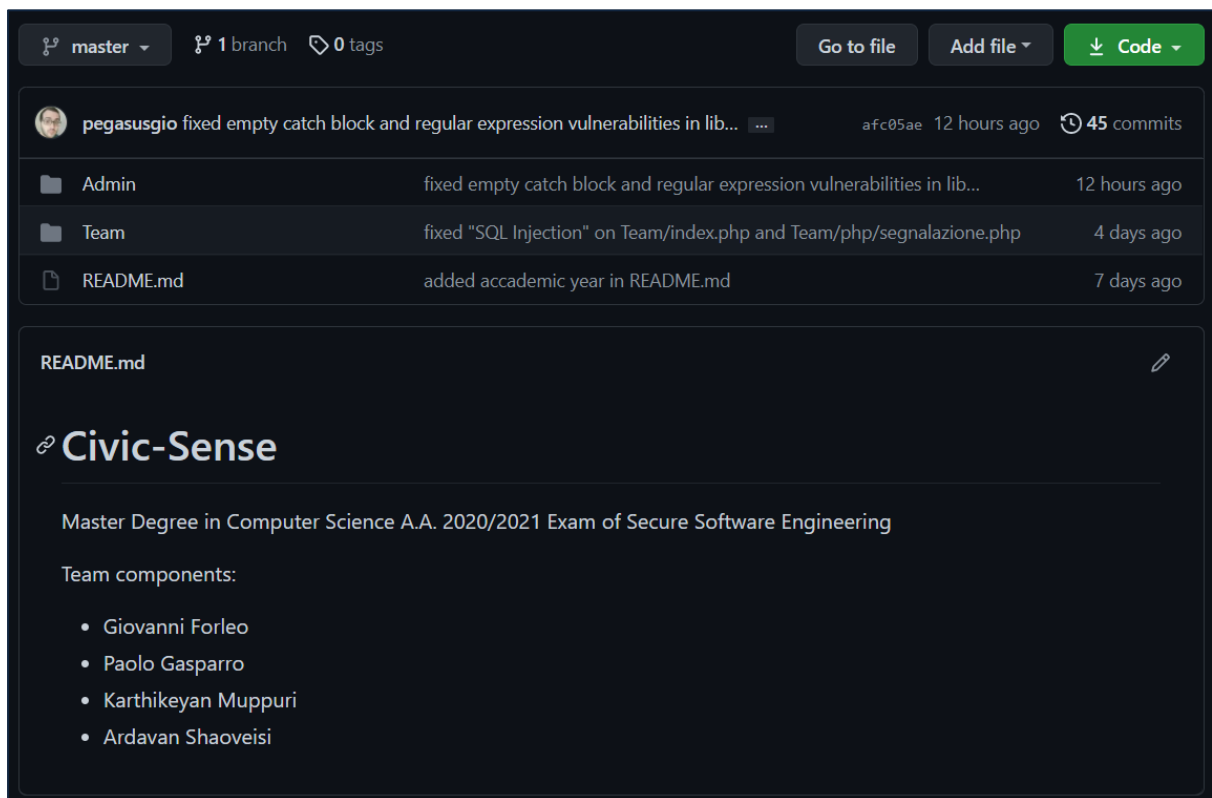


Figure 6: GitHub repository

A reverse-engineering work has been performed: the entire files have been checked to look for the tables which the database had in the original version. So, it has been created, using phpMyAdmin (always from XAMPP), a new database named *civicsense* containing 3 tables:

- *segnalazioni*
- *team*
- *admin*

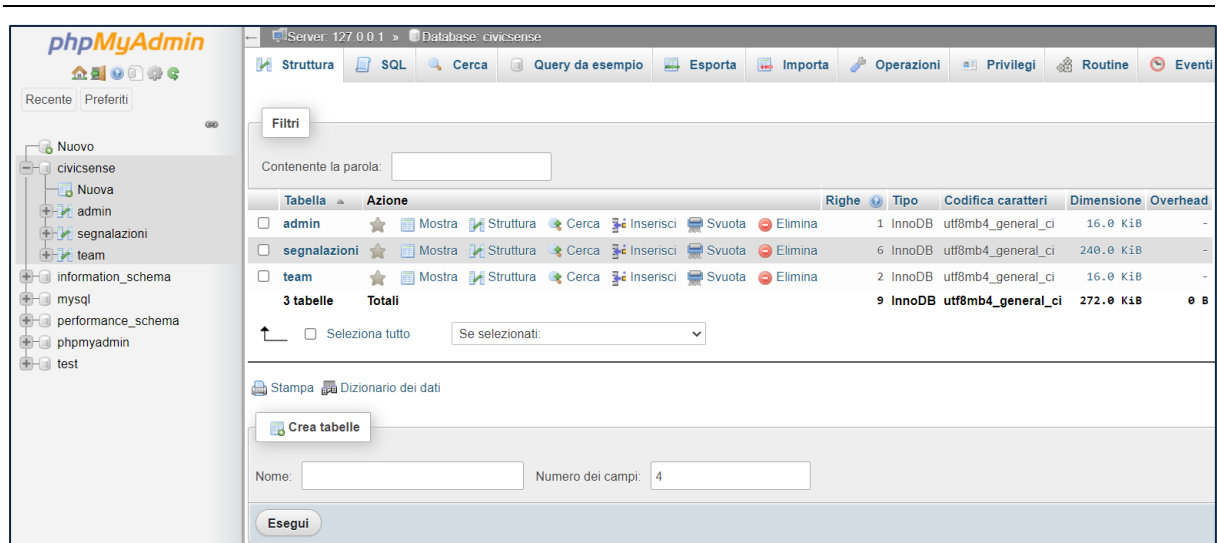


Figure 7: database civicsense

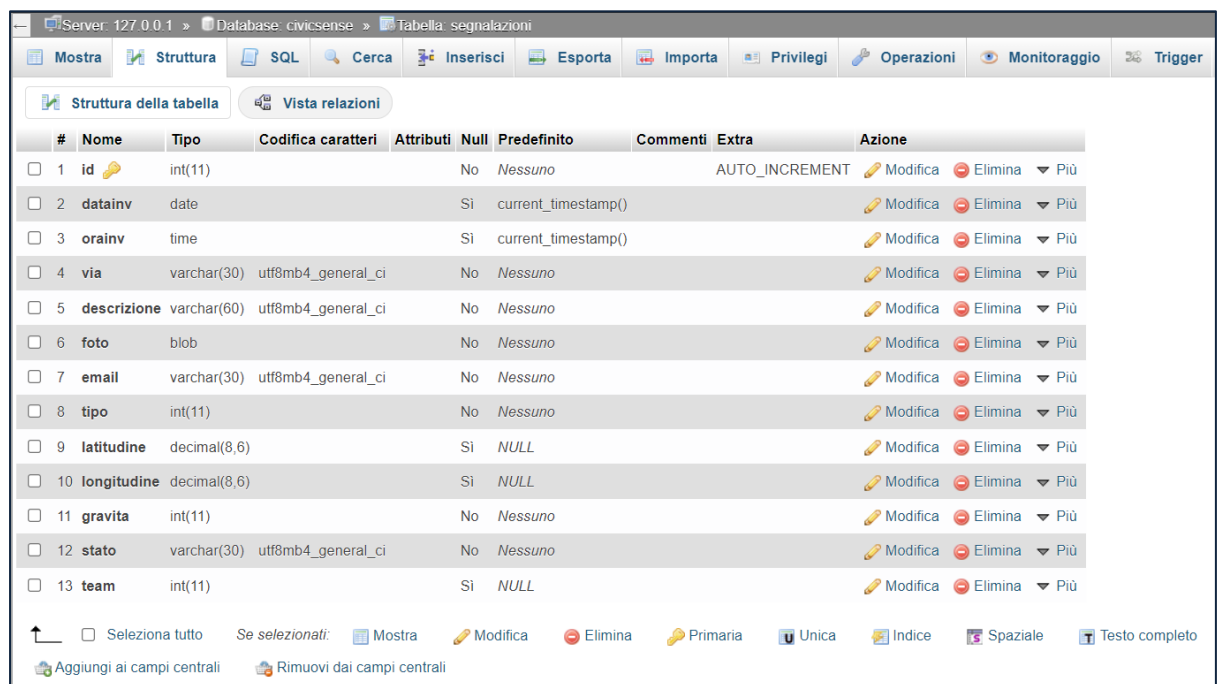


Figure 8: type of fields of the table 'segnalazioni'

Server: 127.0.0.1 » Database: civicsense » Tabella: segnalazioni

Mostra Struttura SQL Cerca Inserisci Esporta Importa Privilegi Operazioni Monitoraggio Trigger

✓ Mostro le righe 0 - 5 (6 del totale. La query ha impiegato 0,0020 secondi.)

SELECT * FROM `segnalazioni`

Profiling [Modifica inline] [Modifica] [Spiega SQL] [Crea il codice PHP] [Aggiorna]

Mostra tutti | Numero di righe: 25 | Filtra righe: Cerca nella tabella | Ordina per chiave: Nessuno

+ Opzioni

			id	datainv	orainv	via	descrizione	foto	email	tipo	latitudine	longitudine	gravita	stato	team
<input type="checkbox"/>	Modifica	Copia	Elimina	1	2021-07-09	11:02:35	via Orazio Flacco	il rosso non si accende	[BLOB - 20.7 KiB]	utente@gmail.com	4	40.382003	17.367155	3	NULL
<input type="checkbox"/>	Modifica	Copia	Elimina	2	2021-07-07	11:37:45	via Torquato Tasso	ci sono rifiuti per strada	[BLOB - 48.2 KiB]	utente1@gmail.com	2	40.382003	17.367155	3	NULL
<input type="checkbox"/>	Modifica	Copia	Elimina	3	2021-07-10	12:06:10	via Giovanni Pascoli	c'è una fossa aperta	[BLOB - 59.1 KiB]	utente2@gmail.com	3	40.055556	17.976389	1	NULL
<input type="checkbox"/>	Modifica	Copia	Elimina	4	2021-07-07	11:47:39	viale della Resistenza	ci sono escrementi di animali sul prato	[BLOB - 14.2 KiB]	utente3@gmail.com	1	40.055556	17.976389	1	NULL
<input type="checkbox"/>	Modifica	Copia	Elimina	5	2021-07-07	12:00:44	via XX Settembre	lampioncino fulminato	[BLOB - 15.2 KiB]	utente4@gmail.com	5	40.531230	17.585220	1	NULL
<input type="checkbox"/>	Modifica	Copia	Elimina	6	2021-07-09	21:58:28	via Roma	lampioncino rotto	[BLOB - 20.1 KiB]	utente5@gmail.com	5	40.949180	17.297170	2	NULL

Seleziona tutto | Se selezionati: Modifica Copia Elimina Esporta

Mostra tutti | Numero di righe: 25 | Filtra righe: Cerca nella tabella | Ordina per chiave: Nessuno

Figure 9: population of the table 'segnalazioni'

Server: 127.0.0.1 » Database: civicsense » Tabella: team

Mostra Struttura SQL Cerca Inserisci Esporta Importa Privilegi Operazioni Monitoraggio Trigger

Struttura della tabella Vista relazioni

#	Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito	Commenti	Extra	Azione
<input type="checkbox"/> 1	codice	int(11)			No	Nessuno		AUTO_INCREMENT	Modifica Elimina Più
<input type="checkbox"/> 2	email_t	varchar(30)	utf8mb4_general_ci		No	Nessuno			Modifica Elimina Più
<input type="checkbox"/> 3	npersone	int(11)			No	1			Modifica Elimina Più
<input type="checkbox"/> 4	nomi	varchar(30)	utf8mb4_general_ci		No	Nessuno			Modifica Elimina Più
<input type="checkbox"/> 5	password	char(24)	utf8mb4_general_ci		No	Nessuno			Modifica Elimina Più

Seleziona tutto | Se selezionati: Mostra Modifica Elimina Primaria Unica Indice Spaziale Testo completo

Aggiungi ai campi centrali Rimuovi dai campi centrali

Figure 10: type of fields of the table 'team'

Server: 127.0.0.1 » Database: civicsense » Tabella: team

Mostra Struttura SQL Cerca Inserisci Esporta Importa Privilegi Operazioni

✓ Mostro le righe 0 - 1 (2 del totale, La query ha impiegato 0,0017 secondi.)

`SELECT * FROM `team``

☐ Profiling [Modifica inline] [Modifica] [Spiega SQL] [Crea il codice PHP] [Aggiorna]

☐ Mostra tutti | Numero di righe: 25 | Filtra righe: Cerca nella tabella | Ordina per chiave: Nessuno

+ Opzioni

		codice	email_t	npersone	nomi	password
<input type="checkbox"/>	Modifica	Copia	Elimina	1	gasparro.paolo@gmail.com	1 Paolo mQ2ZG3FGu+chw7HWvyOdzQ==
<input type="checkbox"/>	Modifica	Copia	Elimina	2	pegasusgio@gmail.com	2 Roberto,Giovanni LrYpQy2+r0J8WBZj2Q6uzQ==

☐ Seleziona tutto | Se selezionati: Modifica Copia Elimina Esporta

☐ Mostra tutti | Numero di righe: 25 | Filtra righe: Cerca nella tabella | Ordina per chiave: Nessuno

Figure 11: population of the table 'team'

Server: 127.0.0.1 » Database: civicsense » Tabella: admin

Mostra Struttura SQL Cerca Inserisci Esporta Importa Privilegi Operazioni Monitoraggio Trigger

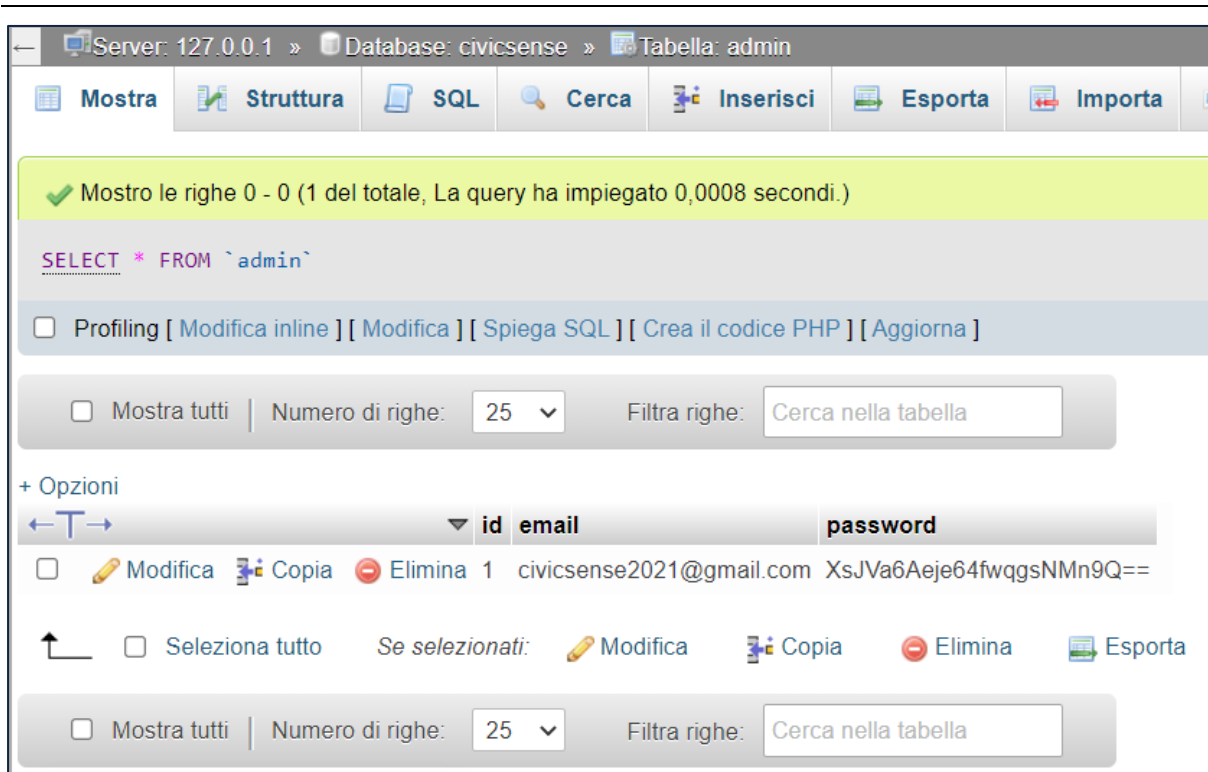
Struttura della tabella Vista relazioni

#	Nome	Tipo	Codifica caratteri	Attributi	Null	Predefinito	Commenti	Extra	Azione
<input type="checkbox"/> 1	id	int(11)			No	Nessuno		AUTO_INCREMENT	Modifica Elimina Più
<input type="checkbox"/> 2	email	varchar(30)	utf8mb4_general_ci		No	Nessuno			Modifica Elimina Più
<input type="checkbox"/> 3	password	char(24)	utf8mb4_general_ci		No	Nessuno			Modifica Elimina Più

☐ Seleziona tutto | Se selezionati: Mostra Modifica Elimina Primaria Unica Indice Spaziale Testo completo

Aggiungi ai campi centrali Rimuovi dai campi centrali

Figure 12: type of fields of the table 'admin'



Server: 127.0.0.1 » Database: civicsense » Tabella: admin

Mostra | Struttura | SQL | Cerca | Inserisci | Esporta | Importa

✓ Mostro le righe 0 - 0 (1 del totale, La query ha impiegato 0,0008 secondi.)

`SELECT * FROM `admin``

☐ Profiling [Modifica inline] [Modifica] [Spiega SQL] [Crea il codice PHP] [Aggiorna]

☐ Mostra tutti | Numero di righe: 25 | Filtra righe: Cerca nella tabella

+ Opzioni

	id	email	password
<input type="checkbox"/> Modifica <input type="checkbox"/> Copia <input type="checkbox"/> Elimina	1	civicsense2021@gmail.com	XsJV6Aeje64fwqgsNMn9Q==

☐ Seleziona tutto | Se selezionati: ☐ Modifica ☐ Copia ☐ Elimina ☐ Esporta

☐ Mostra tutti | Numero di righe: 25 | Filtra righe: Cerca nella tabella

Figure 13: population of the table 'admin'

In the next pages two screenshots, regarding the Team page and one of the five sub-categories of 'Segnalazioni', have been added to show how the system should properly work once it has been fixed.

Area riservata

GESTIONE TEAM

Home

Segnalazioni

Team

Tabella team

Show 10 entries

Search:

CODICE TEAM	E-MAIL	COMPONENTI
1	gasparro.paolo@gmail.com	Paolo
2	pegasusgio@gmail.com	Roberto,Giovanni

Showing 1 to 2 of 2 entries

Previous 1 Next

Segnalazioni senza team

CODICE SEGNALAZIONE	VIA	GRAVITA'	TIPO
1	via Orazio Flacco	3	4
2	via Torquato Tasso	3	2
3	via Giovanni Pascoli	1	3
4	viale della Resistenza	1	1
5	via XX Settembre	1	5
6	via Roma	2	5

Assegna una segnalazione ad un team

CODICE SEGNALAZIONE:

SELEZIONA L'EMAIL DEL TEAM: gasparro.paolo@gmail.com

Invia

Elimina un team

CODICE TEAM DA ELIMINARE:

Invia

Inserisci un nuovo team

E-MAIL TEAM:

NOMI E COGNOMI DEI COMPONENTI:

NUMERO DI COMPONENTI:

PASSWORD:

Invia

Figure 14: Team page

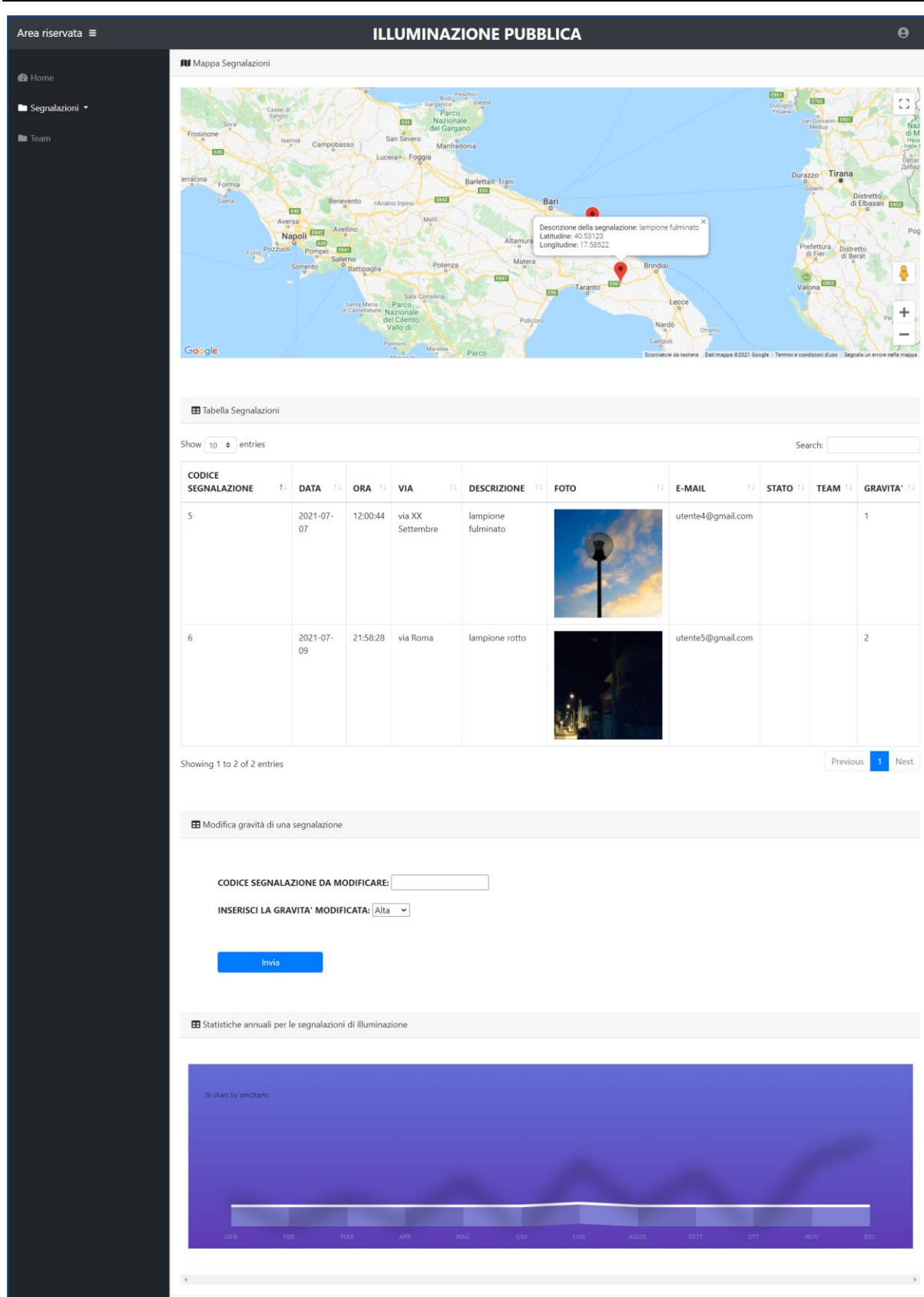


Figure 15: example of fixed and working sub-category (Illuminazione pubblica)

2. VULNERABILITIES ANALYSIS

2.1 Static Code Analysis

Static code analysis has been performed using **Fortify SCA** scan tool. Below will be shown a report containing all the **433** vulnerabilities revealed from the first scan (on the original system) categorized by business risk (Critical, High, Medium, and Low).

Issues by Priority

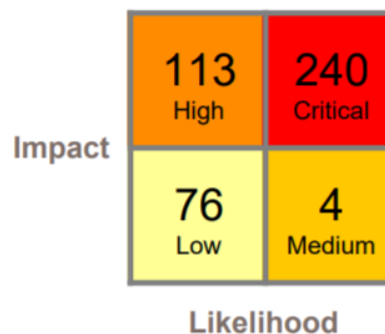


Figure 16: All the vulnerabilities categorized by Business risk.

5 Most Prevalent Critical-Priority Issues

Category	Issues
Cross-Site Scripting: Persistent	165
SQL Injection	39
Privacy Violation	32
Password Management: Hardcoded Password	3
Path Manipulation	1

Issues by Attack Vector

Attack Vector	Issues
Database	169
Network	0
Web	42
Web Service	0
Other	222
Total	433

Figure 17: Two other kind of vulnerabilities categorization

In the next sections, all the vulnerabilities found by Fortify Scan Tool, grouped by Business Risk (Critical, High, Medium, Low), will be presented and described in detail.



2.1.1 Critical vulnerabilities

Critical–priority issues have high impact and high likelihood. Critical–priority issues are easy to detect and exploit and result in large asset damage. These issues represent the highest security risk to the application. As such, they should be remediated immediately, and the team has decided to follow this strategy.

SQL Injection

The initial scan detected **39** vulnerabilities of this kind.

The root cause of a SQL injection vulnerability is the ability of an attacker to change context in the SQL query, causing a value that the programmer intended to be interpreted as data to be interpreted as a command instead.

This vulnerability occurs when there is an invocation of a SQL query built with input that comes from an untrusted source. Exploiting this vulnerability an attacker could manipulate queries to obtain sensitive information about what is stored in the database. For example, in the file ‘Admin/inserisci.php’ it is possible to see this kind of vulnerability.

```
$sql = "INSERT INTO segnalazioni
(datainv, orainv, via, descrizione, foto, email, tipo, latitudine, longitudine)
VALUES
('$data', '$ora', '$via', '$descr', '$foto', '$email', '$tipo', '$lat', '$long') ";
$result = mysqli_query($conn,$sql);
```

Password Management: Hardcoded Password

The initial scan detected **3** vulnerabilities of this kind.

It is never a good idea to hardcode a password. Not only does hardcoding a password allow all the project's developers to view the password, but it also makes fixing the problem extremely difficult. After the code is in production, the password cannot be changed without patching the software. If the account protected by the password is compromised, the owners of the system must choose between security and availability.

For example, in the file ‘Admin\login.php’ it is possible to see this kind of vulnerability.

```
if(isset($_POST['email']) && isset($_POST['password'])){
    $email = $_POST['email'];
    $password = $_POST['password'];
    if($email == "civicsense2019@gmail.com")
    {
        if($password == "admin")
        {
            echo 'Accesso consentito alla sezione riservata';
            echo '<script>window.location.href = "index.php";</script>';
        }
    }
}
```

Privacy Violation

The initial scan detected 32 vulnerabilities of this kind.

Privacy violations occur when:

- Private user information enters the program.
- The data is written to an external location, such as the console, file system, or network.

Security and privacy concerns often seem to compete with each other. From a security perspective, you should record all important operations so that any anomalous activity can later be identified. However, when private data is involved, this practice can create risk.

For example, in the file 'Admin\phpmailer\class.smtp.php' it is possible to see this kind of vulnerability. In this case, this debug function shows sensitive data.

```
switch ($this->Debugoutput) {
    case 'error_log':
        //Don't output, just log
        error_log($str);
}
```



Path Manipulation

The initial scan detected only 1 vulnerability of this kind.

Path manipulation errors occur when the following two conditions are met:

- An attacker can specify a path used in an operation on the file system.
- By specifying the resource, the attacker gains a capability that would not otherwise be permitted.

For example, the program might give the attacker the ability to overwrite the specified file or run with a configuration controlled by the attacker, he names the file in a certain way. For example, in the file 'Admin\InserisciDati.php' it is possible to see this kind of vulnerability.

```
try{  
    move_uploaded_file($_FILES['image']['tmp_name'],$file_path);
```

Cross-Site Scripting: Persistent

The initial scan detected 165 vulnerabilities of this kind.

Cross-site scripting (XSS) vulnerabilities occur when:

- Data enters a web application through an untrusted source. In the case of persistent (also known as stored) XSS, the untrusted source is typically a database or other back-end data store, while in the case of reflected XSS it is typically a web request.
- The data is included in dynamic content that is sent to a web user without validation.

The malicious content sent to the web browser often takes the form of a segment of JavaScript, but can also include HTML, Flash, or any other type of code that the browser executes. The variety of attacks based on XSS is almost limitless, but they commonly include transmitting private data like cookies or other session information to the attacker, redirecting the victim to web content controlled by the attacker, or performing other malicious operations on the user's machine under the guise of the vulnerable site.



```
$upload_path = 'img/';
$quer = mysql_query ("SELECT * FROM segnalazioni WHERE tipo = '4' ");
while($row = mysql_fetch_assoc($quer)) {
    echo "
    <tr>
        <td>".$row['id'].<br></td>
        <td>".$row['datainv'].<br></td>
        <td>".$row['orainv'].<br></td>
        <td>".$row['via'].<br></td>
        <td>".$row['descrizione'].<br></td>
        <td></td>
        <td>".$row['stato'].<br></td>
        <td>".$row['team'].<br></td>
        <td>".$row['gravita'].<br></td>
    </tr> ";
}
```

The data taken from the database and printed in the html body are not trusted data. A malicious code can be injected in the rows.



2.1.2 High vulnerabilities

High-priority issues have high impact and low likelihood. High-priority issues are often difficult to detect and exploit but can result in large asset damage. These issues represent a high security risk to the application. High-priority issues should be remediated in the next scheduled patch release.

Log Forging

The initial scan detected **6** vulnerabilities of this kind.

Log forging vulnerabilities occur when:

- Data enters an application from an untrusted source.
- The data is written to an application or system log file.

Applications typically use log files to store a history of events or transactions for later review, statistics gathering, or debugging. Depending on the nature of the application, the task of reviewing log files may be performed manually on an as-needed basis or automated with a tool that automatically culls logs for important events or trending information.

If the log file is processed automatically, the attacker may be able to render the file unusable by corrupting the format of the file or injecting unexpected characters. For example, in the file 'Admin\phpmailer\class.smtp.php' it is possible to see this kind of vulnerability. In this case, the string '\$str' is not checked and can contain malicious code to inject.

```
switch ($this->Debugoutput) {  
    case 'error_log':  
        //Don't output, just log  
        error_log($str);  
}
```

Password Management: Empty Password

The initial scan detected **83** vulnerabilities of this kind.

It is never a good idea to assign an empty string to a password variable. If the empty password is used to successfully authenticate against another system, then the corresponding account's security is likely compromised because it accepts an empty password. If the empty password is merely a placeholder until a legitimate value can be assigned to the variable, then it can confuse anyone unfamiliar with the code and potentially cause problems on unexpected control flow paths.

For example, in the file 'Admin\login.php' it is possible to see this kind of vulnerability.

```
//Connessione Database  
$conn = mysql_connect ("localhost", "root", "") or die ("Connessione non riuscita");
```

Privacy Violation: Autocomplete

The initial scan detected **4** vulnerabilities of this kind.

With autocompletion enabled, some browsers retain user input across sessions, which could allow someone using the computer after the initial user to see information previously submitted.

For example, in the file 'Admin\login.php' it is possible to see this kind of vulnerability, in fact the input tag does not contain the attribute Autocomplete set to false.

```
<div class="form-group">  
  <div class="form-label-group">  
    <input type="password" id="inputPassword" name="password" class="form-control" placeholder="Password" required="required">  
    <label for="inputPassword"> Password </label>  
  </div>  
</div>
```



Weak Encryption: Inadequate RSA Padding

The initial scan detected 2 vulnerabilities of this kind.

In practice, encryption with an RSA public key is usually combined with a padding scheme. The purpose of the padding scheme is to prevent attacks on RSA that only work when the encryption is performed without padding. For example, in the file 'Admin\phpmailer\class.phpmailer.php' it is possible to see this kind of vulnerability. The method 'dkim_sign()' in 'class.phpmailer.php' performs public key RSA encryption without OAEP padding, thereby making the encryption weak.

```
if (openssl_private_encrypt($eb, $signature, $privKey, OPENSSL_NO_PADDING)) {  
    openssl_pkey_free($privKey);  
    return base64_encode($signature);  
}
```

Access Control: Database

The initial scan detected 2 vulnerabilities of this kind.

Database access control errors occur when:

- Data enters a program from an untrusted source.
- The data is used to specify the value of a primary key in a SQL query.

For example, in the file 'Admin\inserisciDati.php' it is possible to see this kind of vulnerability. Without proper access control, executing a SQL statement that contains a user-controlled primary key can allow an attacker to view unauthorized records.

```
$via = $_POST['via'];  
$descrizione = $_POST['descrizione'];  
$lat = $_POST['latitudine'];  
$lat = floatval($lat);  
$lng = $_POST['longitudine'];  
$lng = floatval($lng);  
  
try{  
    move_uploaded_file($_FILES['image']['tmp_name'],$file_path);  
    $sql = "INSERT INTO `segnalazioni`(`datainv`, `orainv`, `via`,  
    VALUES (CURRENT_DATE,CURRENT_TIME,'" . $via . "','" . $descrizione . "'  
    $result = mysqli_query($conn,$sql);
```

Denial of Service: Regular Expression

The initial scan detected 4 vulnerabilities of this kind.

There is a vulnerability in implementations of regular expression evaluators and related methods that can cause the thread to hang when evaluating regular expressions that contain a grouping expression that is itself repeated. Additionally, any regular expression that contains alternate subexpressions that overlap one another can also be exploited. This defect can be used to execute a Denial of Service (DoS) attack.

For example, in the file 'Admin\phpmailer\class.smtp.php' it is possible to see this kind of vulnerability.

```
if (preg_match("/^([0-9]{3})[ -](?:([0-9]\\.[0-9]\\.[0-9]) )?/", $this->last_reply, $matches)) {
    $code = $matches[1];
    $code_ex = (count($matches) > 2 ? $matches[2] : null);
    // Cut off error code from each response line
    $detail = preg_replace(
       ("/{code}[ -]" .
        ($code_ex ? str_replace('.', '\\.', $code_ex) . ' ' : '') . "/m",
        '',
        $this->last_reply
    );
```

Insecure Randomness

The initial scan detected 12 vulnerabilities of this kind.

Insecure randomness errors occur when a function that can produce predictable values is used as a source of randomness in a security-sensitive context. Computers are deterministic machines, and as such are unable to produce true randomness. Pseudorandom Number Generators (PRNGs) approximate randomness algorithmically, starting with a seed from which subsequent values are calculated.

There are two types of PRNGs: **statistical** and **cryptographic**:

- **Statistical** PRNGs provide useful statistical properties, but their output is highly predictable and form an easy to reproduce numeric stream that is unsuitable for use in cases where security depends on generated values being unpredictable.
- **Cryptographic** PRNGs address this problem by generating output that is more difficult to predict. For a value to be cryptographically secure, it must be

impossible or highly improbable for an attacker to distinguish between the generated random value and a truly random value.

In general, if a PRNG algorithm is not advertised as being cryptographically secure, then it is probably a statistical PRNG and should not be used in security-sensitive contexts, where its use can lead to serious vulnerabilities such as easy-to-guess temporary passwords, predictable cryptographic keys, session hijacking, and DNS spoofing.

For example, in the file 'Admin/vendor/bootstrap/js/bootstrap.bundle.js' it is possible to see this kind of vulnerability.

```
do {  
  // eslint-disable-next-line no-bitwise  
  prefix += ~(Math.random() * MAX_UID); // '  
} while (document.getElementById(prefix));
```

2.1.3 Medium vulnerabilities

Medium-priority issues have low impact and high likelihood. Medium-priority issues are easy to detect and exploit, but typically result in small asset damage. These issues represent a moderate security risk to the application. Medium priority issues should be remediated in the next scheduled product update.

Often Misused: File Upload

The initial scan detected 2 vulnerabilities of this kind.

Regardless of the language in which a program is written, the most devastating attacks often involve remote code execution, whereby an attacker succeeds in executing malicious code in the program's context. If attackers are allowed to upload files to a directory that is accessible from the Web and cause these files to be passed to a code interpreter (e.g., JSP/ASPX/PHP), then they can cause malicious code contained in these files to execute on the server. Even if a program stores uploaded files under a directory that is not accessible from the Web, attackers might still be able to leverage the ability to introduce malicious content into the server environment to mount other attacks. If the program is susceptible to path manipulation, command injection, or dangerous file inclusion vulnerabilities, then an attacker might upload a file with malicious content and cause the program to read or execute it by exploiting another vulnerability.

For example, in the file 'Admin/inserisci.php' it is possible to see this kind of vulnerability. Permitting users to upload files can allow attackers to inject dangerous content or malicious code to run on the server.

```
<form method="post" action="inserisci.php" style="margin-top:5%; margin-left:5%;">
<b>DATA INVIO: <input type="date" name="data"><br><br></b>
<b>ORA INVIO: </b> <input type="time" name="ora"><br><br></b>
<b>VIA (VIA NOMEVIA, N CIVICO, CAP, PROVINCIA (ES: PULSANO O TARANTO), TA, ITALIA: <input type="text" name="via">
<b>DESCRIZIONE: <input type="text" name="descr"><br><br></b>
<b>FOTO: <input type="file" name="foto"><br><br></b>
<b>EMAIL (LA VOSTRA): <input type="email" name="email"><br><br></b>
<b>LATITUDINE: <input type="text" name="lat"><br><br></b>
<b>LONGITUDINE: <input type="text" name="long"><br><br></b>
<b>TIPOLOGIA: </b> <select class="text" name="tipo">
```


Cross-Site Scripting: Poor Validation

The initial scan detected 2 vulnerabilities of this kind.

The use of certain encoding constructs, such as the `<c:out/>` tag with the `escapeXml="true"` attribute (the default behavior), prevents some, but not all cross-site scripting attacks. Depending on the context in which the data appear, characters beyond the basic `<`, `>`, `&`, and `"` that are HTML-encoded and those beyond `<`, `>`, `&`, `"` and `'` that are XML-encoded might take on meta-meaning. Relying on such encoding constructs is equivalent to using a weak deny list to prevent cross-site scripting and might allow an attacker to inject malicious code that will be executed in the browser.

For example, in the file `'Admin/phpmailer/class.phpmailer.php'` it is possible to see this kind of vulnerability.

```
switch ($this->Debugoutput) {  
    case 'error_log':  
        //Don't output, just log  
        error_log($str);  
        break;  
    case 'html':  
        //Cleans up output a bit for a better looking, HTML-safe output  
        echo htmlentities(  
            preg_replace('/[\r\n]+/', '', $str),  
            ENT_QUOTES,  
            'UTF-8'  
        )  
}
```

2.1.4 Low vulnerabilities

Low-priority issues have low impact and low likelihood. Low-priority issues can be difficult to detect and exploit and typically result in small asset damage. These issues represent a minor security risk to the application. Low-priority issues should be remediated as time allows.

Command Injection

The initial scan detected 4 vulnerabilities of this kind.

Command injection vulnerabilities take two forms:

- An attacker can change the command that the program executes: the attacker explicitly controls what the command is;
- An attacker can change the environment in which the command executes: the attacker implicitly controls what the command means.

In this case, we are primarily concerned with the second scenario, in which an attacker can change the meaning of the command by changing an environment variable or by inserting a malicious executable early on the search path.

For example, in the file 'Admin/phpmailer/class.phpmailer.php' it is possible to see this kind of vulnerability.

```
if ($this->SingleTo) {  
    foreach ($this->SingleToArray as $toAddr) {  
        if (!$mail = popen($sendmail, 'w')) {  
            throw new phpmailerException($this->lang('execute') . $this->Sendmail, self::STOP_CRITICAL);  
        }  
    }  
}
```



SQL Injection

The initial scan detected 2 vulnerabilities of this kind.

As already said, this vulnerability occurs when there is an invocation of a SQL query built with input that comes from an untrusted source.

For example, in the file 'Team/index.php' it is possible to see this kind of vulnerability.

```
if(isset($_SESSION['idT'])){\n    $sql = "SELECT * FROM segnalazioni WHERE team = ".$_SESSION['idT'];\n    $result = mysqli_query($conn,$sql);\n}
```

Password Management: Password in Comment

The initial scan detected 20 vulnerabilities of this kind.

It is never a good idea to hardcode a password. Storing password details within comments is equivalent to hardcoding passwords. Not only does it allow all of the project's developers to view the password, but it also makes fixing the problem extremely difficult. After the code is in production, the password is now leaked to the outside world and cannot be protected or changed without patching the software. If the account protected by the password is compromised, the owners of the system must choose between security and availability.

For example, in the file 'Team/php/emailTo.php' it is possible to see this kind of vulnerability.

```
try {\n    $mail->SMTPAuth    = true;                // sblocchi SMTP\n    $mail->SMTPSecure  = "ssl";                // metti prefisso per il serv\n    $mail->Host        = "smtp.gmail.com";     // metti il tuo domino es(gma\n    $mail->Port        = 465;                  // inserisci la porta smtp per\n    $mail->SMTPKeepAlive = true;\n    $mail->Mailer      = "smtp";\n    $mail->Username    = "civicsense18@gmail.com"; // DOMINIO username\n    $mail->Password    = "c1v1csense2019";       // DOMINIO password\n    $mail->AddAddress($_SESSION['email']);\n    $mail->SetFrom("civicsense18@gmail.com");\n}
```



Weak Cryptographic Hash

The initial scan detected **16** vulnerabilities of this kind.

MD2, MD4, MD5, RIPEMD-160, and SHA-1 are popular cryptographic hash algorithms often used to verify the integrity of messages and other data. However, as recent cryptanalysis research has revealed fundamental weaknesses in these algorithms, they should no longer be used within security-critical contexts. Effective techniques for breaking MD and RIPEMD hashes are widely available, so those algorithms should not be relied upon for security. In the case of SHA-1, current techniques still require a significant amount of computational power and are more difficult to implement. However, attackers have found the Achilles' heel for the algorithm, and techniques for breaking it will likely lead to the discovery of even faster attacks.

For example, in the file 'Admin/phpmailer/class.phpmailer.php' it is possible to see this kind of vulnerability.

```
protected function generateId() {  
    return md5(uniqid(time()));  
}
```

Cross-Site Request Forgery

The initial scan detected **22** vulnerabilities of this kind.

A cross-site request forgery (CSRF) vulnerability occurs when:

- A Web application uses session cookies.
- The application acts on an HTTP request without verifying that the request was made with the user's consent.

A nonce is a cryptographic random value that is sent with a message to prevent replay attacks. If the request does not contain a nonce that proves its provenance, the code that handles the request is vulnerable to a CSRF attack (unless it does not change the state of the application). This means a Web application that uses session cookies must take special precautions to ensure that an attacker cannot trick users into submitting bogus requests.

For example, in the file 'Team/index.php' it is possible to see this kind of vulnerability.

```
<form class="d-none d-md-inline-block form-inline ml-auto mr-0 mr-md-3 my-2 my-md-0">
  <ul class="navbar-nav ml-auto ml-md-0">
    <li class="nav-item dropdown no-arrow" >
```

Hardcoded Domain in HTML

The initial scan detected 6 vulnerabilities of this kind.

Including executable content from another web site is a risky proposition. It ties the security of your site to the security of the other site.

For example, in the file 'Team/segnalazioniilluminazione.php' it is possible to see this kind of vulnerability.

```
<div id="chartdiv"></div>
<script src='https://code.jquery.com/jquery-1.11.2.min.js'></script>
```

JavaScript Hijacking: Vulnerable Framework

The initial scan detected 6 vulnerabilities of this kind.

An application may be vulnerable to JavaScript hijacking if it:

- Uses JavaScript objects as a data transfer format.
- Handles confidential data.

Because JavaScript hijacking vulnerabilities do not occur as a direct result of a coding mistake, the Fortify Secure Coding Rulepacks call attention to potential JavaScript hijacking vulnerabilities by identifying code that appears to generate JavaScript in an HTTP response.

For example, in the file 'Admin/vendor/datatables/jquery.dataTables.jsit's possible to see this kind of vulnerability. Applications that use JavaScript notation to transport sensitive data can be vulnerable to JavaScript hijacking, which allows an unauthorized attacker to read confidential data from a vulnerable application.

```
$.ajax( {
  dataType: 'json',
  url: oLanguage.sUrl,
```

System Information Leak: External (Discovered later in the Fortify scans)

An external information leak occurs when system data or debugging information leaves the program to a remote machine via a socket or network connection.

These vulnerabilities are referred to a debug function which tries to print using the built-in 'echo' function, information that can help an adversary to perform an attack.

These vulnerabilities have been showed up by Fortify only after upgrading the PHPMailer library to the latest version (6.5.0).

```
case 'html':  
    //Cleans up output a bit for a better looking, HTML-safe output  
    echo htmlentities(  
        preg_replace('/[\r\n]+/', '', $str),  
        ENT_QUOTES,  
        'UTF-8'  
    ), "<br>\n";  
    break;
```

System Information Leak: Internal (Discovered later in the Fortify scans)

An internal information leak occurs when system data or debug information is sent to a local file, console, or screen via printing or logging.

The program in question might reveal system data or debugging information in 'class.phpmailer.php' with a call to 'fwrite()' function. The information revealed by 'fwrite()' could help an adversary form a plan of attack.

These vulnerabilities have been showed up by Fortify only after upgrading the PHPMailer library to the latest version (6.5.0).

```
$this->edebug("To: {$toAddr}");  
fwrite($mail, 'To: ' . $toAddr . "\n");  
fwrite($mail, $header);  
fwrite($mail, $body);
```

2.2 Security Fix

Here below are showed all the **433** vulnerabilities identified according to the OWASP Top 10 2017 in the initial scan:

Issues by OWASP Top Ten 2017

OWASP Top Ten 2017 Category	Priority			
	Critical	High	Medium	Low
A1 Injection	39	6	2	6
A2 Broken Authentication	0	0	0	0
A3 Sensitive Data Exposure	35	89	0	36
A4 XML External Entities (XXE)	0	0	0	0
A5 Broken Access Control	1	2	0	0
A6 Security Misconfiguration	0	0	0	0
A7 Cross-Site Scripting (XSS)	165	0	2	0
A8 Insecure Deserialization	0	0	0	0
A9 Using Components with Known Vulnerabilities	0	0	0	0
A10 Insufficient Logging and Monitoring	0	0	0	0
None	0	16	0	34

2.2.1 A1 Injection

The vulnerabilities found according to the OWASP Top 10 2017 are **53** (39 Critical, 6 High, 2 Medium and 6 Low).

Critical (39)	High (6)	Medium (2)	Low (6)
SQL Injection (39)	Log Forging (2)	Often Misused: File Upload (2)	Command Injection (4)
			SQL Injection (2)

SQL Injection (Critical)

When a SQL query is constructed, the programmer knows what should be interpreted as part of the command and what should be interpreted as data. Parameterized SQL statements can enforce this behavior by disallowing data-

directed context changes and preventing nearly all SQL injection attacks. Parameterized SQL statements are constructed using strings of regular SQL, but where user-supplied data needs to be included, they include bind parameters, which are placeholders for data that is subsequently inserted. In other words, bind parameters allow the programmer to explicitly specify to the database what should be treated as a command and what should be treated as data. When the program is ready to execute a statement, it specifies to the database the runtime values to use for each of the bind parameters without the risk that the data will be interpreted as a modification to the command.

When connecting to MySQL, the example in the file 'Admin/inserisci.php' showed in the section 2.1.1, can be rewritten using parameterized SQL statements (instead of concatenating user supplied strings) as follows:

```
$query = "INSERT INTO segnalazioni
        (datainv, orainv, via, descrizione, foto, email, tipo, latitudine, longitudine)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?) ";
$stmt = $conn->prepare($query) or die("Connessione non riuscita");
$stmt->bind_param('ssssssss', $data, $ora, $via_c, $descr_c, $foto_c, $email_c, $tipo_c, $lat_c, $long_c);
$result = $stmt->execute();

if ($result) {
    echo "<center> inserimento avvenuto. </center>";
}
```

Furthermore, it has been necessary to replace all the functions with pattern 'mysql_*' with the updated versions 'mysqli_*' (like for example 'mysqli_query' or 'mysqli_connect') because the first one is now deprecated.

Log Forging (Critical)

To prevent log forging attacks with indirection there is the need to create a set of legitimate log entries that correspond to different events that must be logged and only log entries from this set. To capture dynamic content, such as users logging out of the system, always use server-controlled values rather than user-supplied data. This ensures that the input provided by the user is never used directly in a log entry.

To solve this kind of vulnerability that involves an external library PHPMailer (which includes 'class.phpmailer.php' and 'class.smtp.php'), it has been first

necessary to upload the entire library (it has been used the 6.5.0 instead of the outdated 5.2.27).

Then, since the variable can contain malicious code, it is necessary to filter it by using an appropriate method named 'filter_var' which takes 2 arguments: the variable to filter and the type of filter to apply. In this way we are sure, like in the example below, that we are removing the potential malicious characters from the \$str variable.

```
switch ($this->Debugoutput) {  
    case 'error_log':  
        //Don't output, just log  
        error_log(filter_var($str, FILTER_SANITIZE_STRING));  
        break;
```

Often Misused: File Upload (Medium – False Positive)

This vulnerability has been categorized as False Positive. The recommendation is to disable (unless your program specifically requires its users to upload files) the 'file_uploads' option by including the following entry in php.ini:

- file_uploads = 'off'

Most attacks that rely on uploaded content require that attackers be able to supply content of their choosing. Placing restrictions on this content can greatly limit the range of possible attacks.

Although this mechanism prevents attackers from requesting uploaded files directly, it does nothing to mitigate attacks against other vulnerabilities in the program that allow the attacker to leverage uploaded content. The best way to prevent such attacks is to make it difficult for the attacker to decipher the name and location of uploaded files. Such solutions are often program-specific and vary from storing uploaded files in a directory with a name generated from a strong random value when the program is initialized, to assigning each uploaded file a random name and tracking them with entries in a database.

So, to solve this problem, it has been included in the tag the 'accept' to only accept images and has been added a filter to the file uploaded.

```
<b> FOTO: <input type="file" name="foto" accept="image/*"><br><br></b>
```

```
$foto = (isset($_POST['foto'])) ? $_POST['foto'] : null;  
$foto_c = filter_var($foto, FILTER_SANITIZE_STRING);
```

Command Injection (Low)

This vulnerability can be solved by a future upgrade of PHPMailer library.

SQL Injection (Low)

This kind of vulnerabilities have been solved in the same way have been solved in the same way as for the SQL Injection categorized as Critical. For example, in 'Team/php/segnalazione.php' the final fixed version of the code is the following:

```
$query = "SELECT * FROM segnalazioni WHERE stato <> 'Risolto' AND team = ?";  
$stmt = $conn->prepare($query);  
$stmt->bind_param('s', $_SESSION['idT']);  
$result_query = $stmt->execute();  
  
if ($result_query) {  
    $result = $stmt->get_result();  
  
    while ($row = mysqli_fetch_assoc($result)) {
```

2.2.2 A2 Broken Authentication

There are no vulnerabilities found by the Fortify SCA scan tool belonging to this category.



2.2.3 A3 Sensitive Data Exposure

The vulnerabilities found according to the OWASP Top 10 2017 are **160** (35 Critical, 89 High and 36 Low).

Critical (35)	High (89)	Medium (0)	Low (36)
Privacy Violation (32)	Weak Encryption: Inadequate RSA Padding (2)		Password Management: Password in Comment (20)
Password Management: Hardcoded Password (3)	Privacy Violation: Autocomplete (4)		Weak Cryptographic Hash (16)
	Password Management: Empty Password (83)		

Privacy Violation (Critical)

These vulnerabilities have not been solved and a possible solution will be given in the section 3.2.1.

Password Management: Hardcoded Password (Critical)

Passwords should never be hardcoded and should generally be obfuscated and managed in an external source. Storing passwords in plain text anywhere on the system allows anyone with sufficient permissions to read and potentially misuse the password.

The team decided to use the method 'openssl_encrypt' to encrypt the password. This method takes three parameters: the first is the password to be encrypted, the second is the encryption algorithm and the third is the secret key to be used for the computation.

Obviously, the password in the database needs to be previously put in the encrypted way, allowing the server to decrypt it (with the secret key in the config file) and make the comparison to check if it is equal to the password inserted into the form.

```
$var = parse_ini_file("config.ini");

$query = "SELECT * FROM admin where email = ? AND password = ?";
$stmt = $conn->prepare($query);
$stmt->bind_param('ss', $email, openssl_decrypt($password, "AES-128-ECB", $var['SECRETKEY']));
$result_query = $stmt->execute();
```

To reach this goal, it has been created a '**config.ini**' file containing the secret key that must be used to decrypt the password.

```
Admin > ≡ config.ini
1  [First Section]
2  SECRETKEY = 'chiavesupersegreta'
```

Weak Encryption: Inadequate RSA Padding (High)

This vulnerability has been solved upgrading the library **PHPMailer** to the latest version, upgrading the older 5.2.27 version to the newest 6.5.0.

Privacy Violation: Autocomplete (High)

The solution is to explicitly disable autocompletion on forms or sensitive inputs, adding the parameter '**autocomplete**' and setting it '**off**'. By disabling autocompletion, information previously entered will not be presented back to the user as they type.

```
<input type="password" id="inputPassword" name="password" class="form-control" placeholder="Password" required="required" autocomplete="off">
<label for="inputPassword"> Password </label>
```

Password Management: Empty Password (High)

Always read stored password values from encrypted, external resources and assign password variables meaningful values. Ensure that sensitive resources are never protected with empty or null passwords.

By the way, even if using an empty password, using the updated constructor 'mysqli' (available since the 5th version of PHP) instead of the outdated 'mysql', solves this vulnerability probably because it adds a security layer which is transparent to the user.

```
$conn = new mysqli("localhost", "root", "", "civicsense");
```

This kind of problem became more serious if the database is stored on a remote server and not locally as in this case where it was only useful to make the system work and then make some tries on it.

Password Management: Password in Comment (Low)

To fix this vulnerability, the team just remove all the passwords in comment.

Weak Cryptographic Hash (Low)

Discontinue the use of MD2, MD4, MD5, RIPEMD-160, and SHA-1 for data-verification in security-critical contexts. Currently, SHA-224, SHA-256, SHA-384, SHA-512, and SHA-3 are good alternatives.

With the updated version of the PHPMailer library, a lot of these vulnerabilities disappeared, since the weak cryptographic hash **md5** used in the older version (the 5.2.27) has been placed by the more secure **sha-256**.

However, there was still a method using the old md5 hashing algorithm, the one shown below:



```
/**
 * Calculate an MD5 HMAC hash.
 * Works like hash_hmac('md5', $data, $key)
 * in case that function is not available.
 *
 * @param string $data The data to hash
 * @param string $key The key to hash with
 *
 * @return string
 */
protected function hmac($data, $key)
{
    if (function_exists('hash_hmac')) {
        return hash_hmac('md5', $data, $key);
    }
}
```

The solution is to replace the parameter **md5** (that is the algorithm the function uses) chosen with a more secure one, show in the image below.

```
/**
 * Calculate an SHA-512 HMAC hash.
 *
 * @param string $data The data to hash
 * @param string $key The key to hash with
 *
 * @return string
 */
protected function hmac($data, $key)
{
    return hash_hmac('sha512', $data, $key);
}
```



2.2.4 A4 XML External Entities (XXE)

There are no vulnerabilities found by the Fortify SCA scan tool belonging to this category.

2.2.5 A5 Broken Access Control

The vulnerabilities found according to the OWASP Top 10 2017 are **3** (1 Critical and 2 High).

Critical (1)	High (2)	Medium (0)	Low (0)
Path Manipulation (1)	Access Control: Database (2)		

Path Manipulation (Critical)

The best way to prevent path manipulation is with a level of indirection: create a list of legitimate values from which the user must select. With this approach, the user-provided input is never used directly to specify the resource name.

The solution adopted on the 'inserisciDati.php' file is based on the sanitization of both the 'tmp_name' and 'name' fields of the image uploaded.

First, there is a check to verify if file uploaded is an image. After that, the extension of the file is extracted, and the filename is sanitized in order to remove all the potential malicious characters (like '../').

```
$filename = $_FILES['image']['name'];
$filetype = $_FILES['image']['type'];
$filetmp_name = filter_var($_FILES['image']['tmp_name'], FILTER_SANITIZE_STRING);

//check if the type is image
if (str_contains($filetype, "image")) {

    //save the extension
    $ext = pathinfo($filename, PATHINFO_EXTENSION);

    //remove the extension
    $without_extension = pathinfo($str, PATHINFO_FILENAME);

    //clean the name
    $cleaned_file = preg_replace("/[^a-zA-Z0-9]+/", "", $without_extension);
```

Access Control: Database (High)

Rather than relying on the presentation layer to restrict values submitted by the user, access control should be handled by the application and database layers. Under no circumstances should a user be allowed to retrieve or modify a row in the database without the appropriate permissions. Every query that accesses the database should enforce this policy, which can often be accomplished by simply including the current authenticated username as part of the query.

This problem has been fixed in the same way we fixed the SQL Injection vulnerabilities.

2.2.6 A6 Security Misconfiguration

There are no vulnerabilities found by the Fortify SCA scan tool belonging to this category.



2.2.7 A7 Cross-Site Scripting (XSS)

The vulnerabilities found according to the OWASP Top 10 2017 are **167** (165 Critical and 2 Medium).

Critical (165)	High (0)	Medium (2)	Low (0)
Cross-Site Scripting: Persistent (165)		Cross-Site Scripting: Poor Validation (2)	

Cross-Site Scripting: Persistent (Critical)

The solution to XSS is to ensure that validation occurs in the correct places and checks are made for the correct properties.

Because XSS vulnerabilities occur when an application includes malicious data in its output, one logical approach is to validate data immediately before it leaves the application. However, because web applications often have complex and intricate code for generating dynamic content, this method is prone to errors of omission (missing validation). An effective way to mitigate this risk is to also perform input validation for XSS.

For all the instances of this vulnerability the solution adopted is the same: using the method `filter_var` (already explained in the section 2.2.1), setting the specific parameter `FILTER_SANITIZE_*`. In this way, all the possible malicious code is removed.

```

mysqli_select_db($conn, "civicsense") or die("DataBase non trovato"); #connessione al db

$query = mysqli_query($conn, "SELECT * FROM segnalazioni WHERE tipo = '4' ");

while ($row = mysqli_fetch_assoc($query)) {
    echo "
    <tr>
        <td>" . filter_var($row['id'], FILTER_SANITIZE_NUMBER_INT) . " <br></td>
        <td>" . filter_var($row['datainv'], FILTER_SANITIZE_STRING) . " <br></td>
        <td>" . filter_var($row['orainv'], FILTER_SANITIZE_STRING) . " <br></td>
        <td>" . filter_var($row['via'], FILTER_SANITIZE_STRING) . " <br></td>
        <td>" . filter_var($row['descrizione'], FILTER_SANITIZE_STRING) . " <br></td>
        <td><img width='200' height='200' src=data:image/jpeg;base64," . filter_var(base64_encode($row['foto']), FILTER_SANITIZE_STRING)
        <td>" . filter_var($row['email'], FILTER_SANITIZE_EMAIL) . " <br></td>
        <td>" . filter_var($row['stato'], FILTER_SANITIZE_STRING) . " <br></td>
        <td>" . filter_var($row['team'], FILTER_SANITIZE_NUMBER_INT) . " <br></td>
        <td>" . filter_var($row['gravita'], FILTER_SANITIZE_NUMBER_INT) . " <br></td>
    </tr> ";
}

```

Cross-Site Scripting: Poor Validation (Medium)

This vulnerability has not been solved upgrading the library **PHPMailer** to the latest version, passing from the older 5.2.27 to the 6.5.0.

To fix it, the team used the same **filter_var** function used above.

```

switch ($this->Debugoutput) {
    case 'error_log':
        //Don't output, just log
        error_log(filter_var($str, FILTER_SANITIZE_STRING));
        break;
    case 'html':
        //Cleans up output a bit for a better looking, HTML-safe output
        echo htmlentities(
            preg_replace('/[\r\n]+/', '', $str),
            ENT_QUOTES,
            'UTF-8'
        ), "<br>\n";
        break;
}

```

2.2.8 A8 Insecure Deserialization

There are no vulnerabilities found by the Fortify SCA scan tool belonging to this category.



2.2.9 A9 Using Components with Known Vulnerabilities

There are no vulnerabilities found by the Fortify SCA scan tool belonging to this category.

2.2.10 A10 Insufficient Logging and Monitoring

There are no vulnerabilities found by the Fortify SCA scan tool belonging to this category.

2.2.11 Not Set

The vulnerabilities not categorized by the OWASP Top 10 2017 are **50** (16 High and 34 Low).

Critical (0)	High (16)	Medium (0)	Low (34)
	Insecure Randomness (12)		JavaScript Hijacking: Vulnerable Framework (6)
	Denial of Service: Regular Expression (4)		Hardcoded Domain in HTML (6)
			Cross-Site Request Forgery (22)

Cross-Site Request Forgery (Low)

These vulnerabilities have not been solved and a possible solution will be given in the section 3.2.3.



Insecure Randomness (High – false positive)

Fortify scan tool categorized “Insecure Randomness” as a High Vulnerability but it doesn’t appear in any category of the OWASP Top 10 2017 (so it appears as “Not Set”). The instances found are 13: 1 is referred to the ‘class.phpmailer.php’ file, 4 to ‘jquery.slim.min.js’, 4 to ‘jquery.slim.js’, 2 to ‘bootstrap.bundle.js’ and 2 to ‘bootstrap.bundle.min.js’.

This vulnerability has been categorized as false positive because the random number generator implemented by ‘random()’ function cannot withstand a cryptographic attack, but in our case this vulnerability is detected in three 3rd-party libraries, such as Bootstrap.js, JQuery.js and PHPMailer. In this case it’s a false positive because the usage of random numbers in these two libraries doesn’t need to be cryptographically secure.

Denial of Service: Regular Expression (High)

To fix these vulnerabilities, it should not be allowed untrusted data to be used as regular expression patterns.

Even if this vulnerability depends on a 3rd-party library (PHPMailer), and so in the future upgrades this should be fixed by the releaser, a possible fix consists in filtering the variables with the ‘filter_var’ function and passing as 2nd argument of the function in the first case ‘FILTER_SANITIZE_NUMBER_INT’ and in the second case ‘FILTER_SANITIZE_STRING’ as shown below:

```
if (preg_match('/^([\d]{3})[ -](?:([\d]\.\.[\d]\.\.[\d]{1,2}) )?/', $this->last_reply, $matches)) {
    $code = filter_var(((int) $matches[1]), FILTER_SANITIZE_NUMBER_INT);
    $code_ex = filter_var((count($matches) > 2 ? $matches[2] : null), FILTER_SANITIZE_STRING);
    //Cut off error code from each response line
    $detail = preg_replace(
        "/{ $code }[ -]" .
        ($code_ex ? str_replace('.', '\\.', $code_ex) . ' ' : '') . '/m',
        '',
        $this->last_reply
    );
}
```

JavaScript Hijacking: Vulnerable Framework (Low)

These vulnerabilities have not been solved and a possible solution will be given in the section 3.2.2.

Hardcoded Domain in HTML (Low)

The suggestion for these vulnerabilities is to Keep control over the code that web pages invoke. Scripts or other artifacts from 3rd-party sites should not be included. Then, since the bootstrap and jquery 3rd-party libraries are physically part of the project and they can be found in the 'Admin/vendor/*' folder, it is useless to include a script with an external web source. That is to solve this problem, it was just necessary to include these local paths as source where needed. An example is showed below:

```
<!-- Bootstrap core CSS-->
<link href="../../Admin/vendor/bootstrap/css/bootstrap.min.css" rel="stylesheet">

<!-- Custom fonts for this template-->
<link href="../../Admin/vendor/fontawesome-free/css/all.min.css" rel="stylesheet" type="text/css">

<!-- Page level plugin CSS-->
<link href="../../Admin/vendor/datatables/dataTables.bootstrap4.css" rel="stylesheet">

<!-- Custom styles for this template-->
<link href="../../Admin/css/sb-admin.css" rel="stylesheet">

<!-- grafico -->
<link rel="stylesheet" href="../../Admin/css/graficostyle.css">
```

System Information Leak: External (Low – Discovered later in the Fortify scans)

The solution is to write error messages with security in mind. In production environments, turn off detailed error information in favor of brief messages. Restrict the generation and storage of detailed output that can help administrators and programmers diagnose problems. Debug traces can sometimes appear in non-obvious places (embedded in comments in the HTML for an error page, for example).

Even brief error messages that do not reveal stack traces or database dumps can potentially aid an attacker. For example, an "Access Denied" message can reveal that a file or user exists on the system.

System Information Leak: Internal (Low – Discovered later in the Fortify scans)

The solution is the same as in the case written above (System Information Leak: External).

2.2.12 Final report

As showed below the final scan after all the possible fixes, reported **46** problems splitted in the following way:

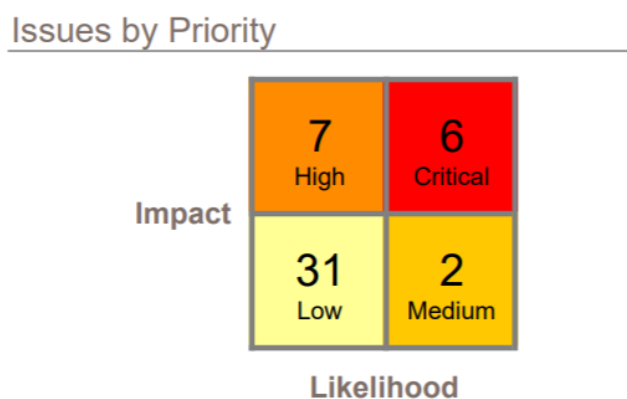


Figure 18: All the vulnerabilities categorized by Business risk.

5 Most Prevalent Critical-Priority Issues

Category	Issues
Privacy Violation	6

Issues by Attack Vector

Attack Vector	Issues
Database	0
Network	0
Web	0
Web Service	0
Other	46
Total	46

Figure 19: Figure 17: Two other kind of vulnerabilities categorization



Issues and Fortify Security Rating by Date

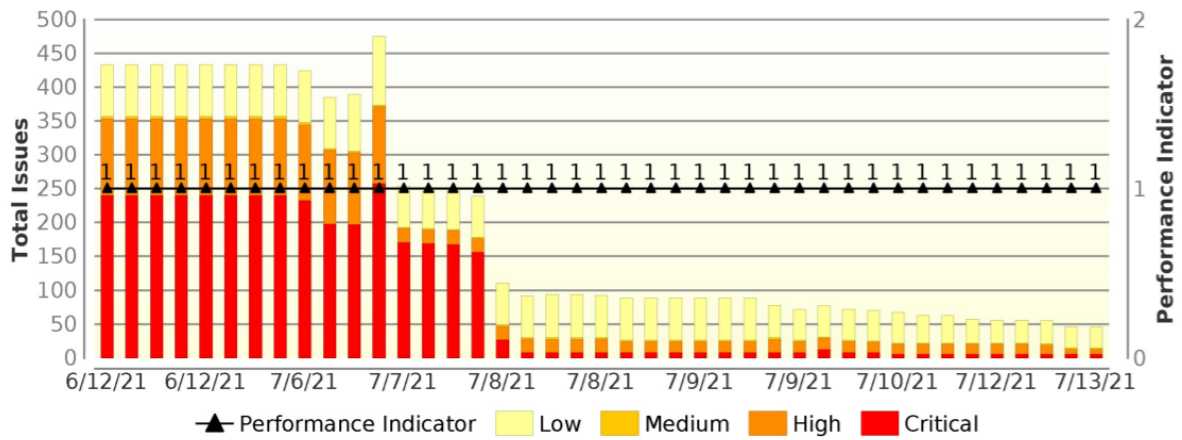


Figure 20: Security Rating by Date

Here below there is instead the final situation from the OWASP Top Ten 2017 categorization. As the report above, there is a total of **46** issues: 4 in A1 Injection, 6 in A3 Sensitive Data Exposure and 36 in Not Set category.

Issues by OWASP Top Ten 2017

OWASP Top Ten 2017 Category	Priority			
	Critical	High	Medium	Low
A1 Injection	0	0	2	2
A2 Broken Authentication	0	0	0	0
A3 Sensitive Data Exposure	6	0	0	0
A4 XML External Entities (XXE)	0	0	0	0
A5 Broken Access Control	0	0	0	0
A6 Security Misconfiguration	0	0	0	0
A7 Cross-Site Scripting (XSS)	0	0	0	0
A8 Insecure Deserialization	0	0	0	0
A9 Using Components with Known Vulnerabilities	0	0	0	0
A10 Insufficient Logging and Monitoring	0	0	0	0
None	0	7	0	29

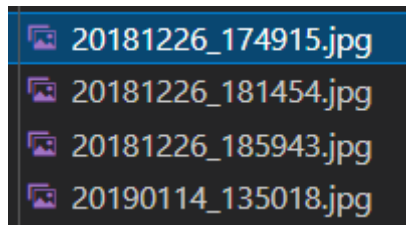
Figure 21: Final report (OWASP Top Ten 2017)

2.3 Vulnerabilities not identified by Fortify

In this section there are all the vulnerabilities that Fortify SCA tool has not identified, but the team has.

2.3.1 A3 Sensitive Data Exposure

The system has an image folder containing possible sensitive information. The data should be stored encrypted, and then decrypted before to be used. Moreover, the names of the files suggest the date in which that image has been taken. For example, in the case of Civic-Sense, inside the 'Admin' folder there is another one called 'img' which contains the following files:



2.3.2 A5 Broken Access Control

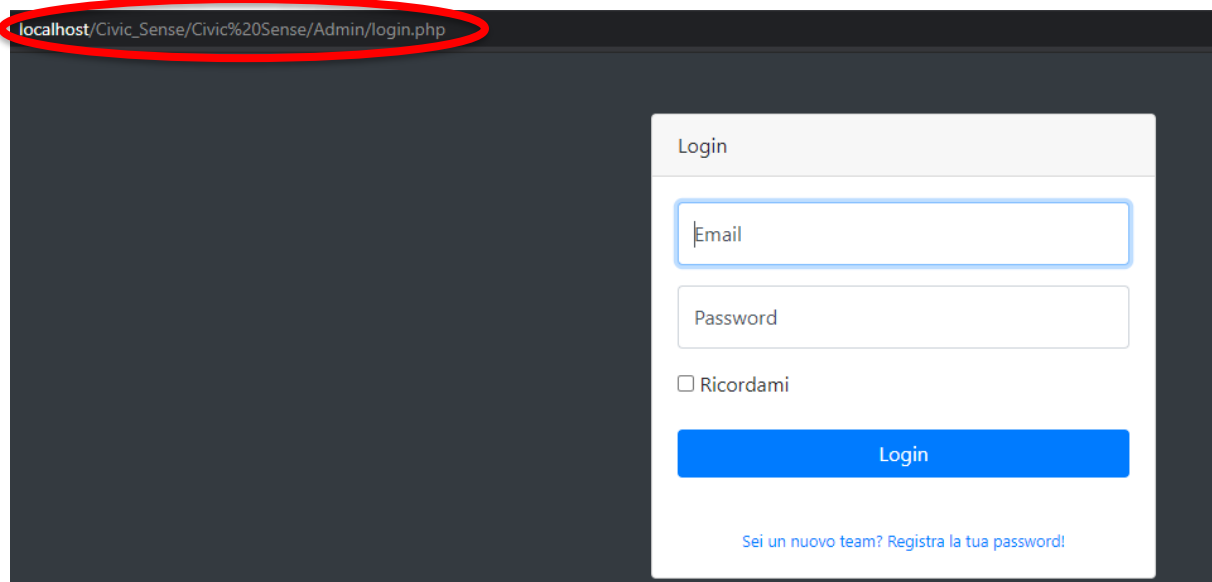
- 1) In 'Admin/login.php' file there is an if statement that contains a wrong check condition: if a user inserts the correct mail OR the correct password (not both), can access into the system.

```
if($password != $row["password"] || $email != $row["email_t"]){
{
    //CODICE JAVASCRIPT
    echo 'ATTENZIONE: La password o la email inserita non è corretta!';
}
else if ($password == $row["password"] || $email == $row["email_t"]){
$_SESSION['email']=$email;
$_SESSION['pass']=$password;
$_SESSION['idT']=$row['codice'];
echo 'Accesso consentito area riservata (TEAM)';
```


2) The file 'Admin/inserisci.php' contains hardcoded password of the database:

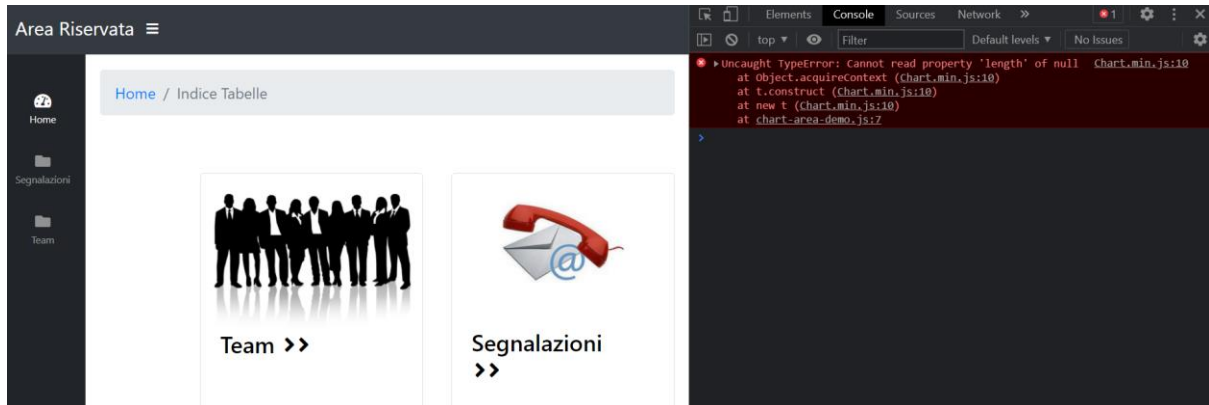
```
$conn = mysqli_connect("localhost", "id8503350_civicsense", "civicsense", "id8503350_civicsense")
```

3) During the login process, and without knowing any kind of credential (mail or password), it is possible for everybody to access to the private area (Admin) just removing 'login.php' from the URL above. This is because there are no session variables properly set.



4) In the main page of the system, if an attacker uses the inspect element mode, can see the path in the server which contains the error shown.

5)



This error is due to the library Chart.js and it disappeared after the update from the version 2.7.2 to the version 3.4.1 of the library.

6) When trying to logout after logging-in as Team, the system redirects to a non-existing page 'login.php' so it gives an error. This is due to the fact that in the file 'Team/index.php' there is a wrong href tag value which redirects to a non-existing file (login.php).

To fix this unexpected behavior it has been changed the value of the href tag with '.../Admin/login.php'.



2.3.3 A9 Using components with Known Vulnerabilities

- 1) In the project there are some dwsync.xml files. If an attacker exploits these files, can retrieve important information:
 - the fact that the server has been implemented using Adobe Dreamweaver and he can try to exploit some possible vulnerability of that software.
 - the path of the files described in the file. Below it is shown an example:

```
C:\Civic_Sense > Civic_Sense > Admin > _notes > dwsync.xml
1  <?xml version="1.0" encoding="utf-8" ?>
2  <dwsync>
3  <file name="index.php" server="C:/xampp/htdocs/Ingegneria/" local="131917859245487473" remote="131917859240000000" Dst="1" />
4  <file name="segnalazionilluminazione.php" server="C:/xampp/htdocs/Ingegneria/" local="131931674580000000" remote="131931674580000000" Dst="1" />
5  <file name="segnalazionii.php" server="C:/xampp/htdocs/Ingegneria/" local="131897698145104583" remote="131897698140000000" Dst="1" />
6  <file name="team.php" server="C:/xampp/htdocs/Ingegneria/" local="131924566620000000" remote="131924566620000000" Dst="1" />
7  <file name="segnalazioniverde.php" server="C:/xampp/htdocs/Ingegneria/" local="131932475305117715" remote="131932475300000000" Dst="1" />
8  <file name="login.php" server="C:/xampp/htdocs/Ingegneria/" local="131918710417615668" remote="131918710410000000" Dst="1" />
9  <file name="InserisciDati.php" server="C:/xampp/htdocs/Ingegneria/" local="131919453443682869" remote="131919453440000000" Dst="1" />
10 </dwsync>
```

This vulnerability is called 'Adobe Dreamweaver dwsync.xml Remote Information Disclosure'. As said before Adobe's Dreamweaver is known to produce 'dwsync.xml' files. These contain synchronization information that may include the list of files and directories synchronized. This can lead to information disclosure.

A possible solution is to disable the 'Maintain synchronization information' option from the Remote Info category of the advanced view of the Site Definition dialog box. In addition, remove the offending files if already created by the system.

- 2) There are two file that cannot be reached browsing the entire system ('Admin/inserisci.php' e 'Admin/php/loginn.php'). However, if an attacker uses some scanning tool can find these files and use them to retrieve some sensitive data.



2.3.4 Observations

Looking at the project and analyzing its structure it's possible to see that has been designed with two main folders: 'Admin' and 'Team'. Each one of these folders contain the main 'index.php', other php files and the main libraries mentioned till now (like PHPMailer, Bootstrap or JQuery). What emerged is that is useless to have the same libraries both in Admin and in Team, so the same files two times, and this is symptom of bad software designing phase. Indeed, it is a good practice to have a unique folder for each library and create only the necessary php files needed.

For example, if we look into the 'Team' folder we can see that it depends on the 'Admin' one because of the login page ('Admin/login.php'). This logic should not be separated but, instead, in the 'Admin/login.php' should have been included a tab or a section for the Admin login and another one for the Team login. The idea of separation of Admin and Team logic is a good idea, but it has been implemented in the wrong way and it should be implemented as really two separate projects with no dependencies one from the other: in this way a full decoupling can be obtained.

Moreover, Fortify scan tool, due to its logic, identified all the vulnerabilities referred to the libraries twice: this because of course, as said before, we have the same PHPMailer library both in Admin and in Team folders.

To solve this kind of behavior, the libraries in the Team folder have been deleted and in the files they were required, the path has been changed with the one referring to the Admin so to maintain only one copy of the libraries.

3. PRIVACY ANALYSIS

The first part of privacy analysis involves the identification of privacy design strategies for system re-engineering.

The privacy patterns that implement the strategies can be found on the website <https://privacypatterns.org/patterns/>.

Privacy Patterns have also been used to explain the unresolved vulnerabilities. This part will be explained in detail later in this chapter.

Then has been provided a Target architecture of the system (component diagram) which integrates the privacy patterns.

At last, it will be provided a description on the architecture of the system and the way through which the latter uses the Privacy Patterns that implement the Privacy Design Strategies.

3.1 Privacy Assessment

In the Privacy Assessment, the vulnerabilities identified during the static code analysis in the section 2 are provided as input to the Privacy Knowledge Base to identify:

- The principles of Privacy by Design violated by vulnerabilities;
- The Privacy Design Strategies to be implemented in the system to respect the principles of Privacy by Design;
- The privacy patterns that substantiate the Privacy Design Strategies.

The results of this analysis are reported in the Privacy Report.

A design strategy describes a way to achieve a certain design goal with certain properties that distinguish it from other (basic) approaches for achieving the same goal. There are 8 Privacy Design Strategies, based on the legal perspective of privacy, divided in two different categories:

- **Data-Oriented Strategies** (focus on the privacy-friendly processing of the data):
 - Minimize: Limit the processing of personal data as much as possible;
 - Hide: Protect personal data or make it unobservable. Make sure it does not become public or known;

- **Separate:** Separate the processing of personal data as much as possible;
- **Abstract:** Limit the detail in which personal data is processed.
- **Process-Oriented Strategies** (focus on the processes surrounding the responsible handling of personal data):
 - **Inform:** Inform data subjects about the processing of their personal data in a timely and adequate manner;
 - **Control:** Provide data subjects an adequate control over the processing of their personal data;
 - **Enforce:** Commit to processing personal data in a privacy-friendly way, and adequately enforce this;
 - **Demonstrate:** Demonstrate personal data is being processed in a privacy-friendly manner.

3.1.2 Privacy Pattern applied

Here below there is the list of the Privacy Pattern applied to the re-engineering of the system.

Personal Data Store

This pattern has been used both in Data and Process Oriented Process Design since it implements both Design strategies **Separate** and **Control**.

It is used to let the subjects keep control on their personal data that are stored on a personal device. The main purpose is to combine a central server and secure personal tokens. Personal tokens, which can take the form of USB keys, embed a database system, a local web server and a certificate for their authentication by the central server. In this way data subjects can decide on the status of their data and, depending on their level of sensitivity, choose to record them exclusively on their personal token or to have them replicated on the central server.

Encryption with user-managed keys

This pattern has been used both in Data and Process Oriented Process Design since it implements both Design strategies: **Hide** and **Control**.



It can be used by a user who wants to store or transfer their personal information through an online service while ensuring their privacy and specifically preventing unauthorized access to their personal information.

Anonymous reputation-based Blacklist

This pattern implements the Design Strategy **Hide**.

Suppose to have a service provider that wants to track an anonymous-authenticated-user behavior during the usage of the system.

The service provider can assign a reputation score to its users, based on their interactions with the service. Those who misbehave earn a bad reputation, and they are added to a blacklist and banned from using the service anymore. However, these scoring systems traditionally require the user identity to be disclosed and linked to their reputation score; hence, they conflict with anonymity.

To solve this conflict, the Anonymous reputation-based Blacklist pattern implements the following strategy: every time an authenticated user holds a session at the service, the service provider assigns and records a reputation value for that session, depending on the user behavior during the session.

Metadata Manager

This pattern implements the Design Strategy **Minimize**.

The module acts when a user shares documents or web resources with 3rd-party services; it allows users who are not fully aware of the attached metadata to manage them and then delete them.

Location Granularity

This pattern implements the Design Strategy **Abstract**.

Collecting more information than is necessary can harm the user's privacy and increase the risk for the service (in the case of a security breach, for example), but location data may still need to be collected to provide the service.



By collecting or distributing only the necessary level of granularity, a service may be able to maintain the same functionality without requesting or distributing potentially sensitive data.

Onion Routing

This pattern implements the Design Strategy **Hide**.

This pattern provides unlinkability between senders and receivers by encapsulating the data in different layers of encryption, limiting the knowledge of each node along the delivery path.

Unusual Activities

This pattern implements the Design Strategy **Inform**.

In case of a suspicious activity this pattern alerts the user and use multi-factor authentication (or a token-based authentication) to verify the identity.

If location is provided, for example, it may hint at unlikely account activity. The authenticated user should be able to review and take further action.

Data Breach Notification Pattern

This pattern implements the Design Strategy **Inform**.

When data breaches occur, numerous risks become apparent for multiple parties, these parties need to be notified and the risks need to be mitigated.

The main purpose of this pattern is to detect and react to data breaches quickly, notifying the supervisory authority of details, particularly risk mitigation, in order to establish whether users must also be informed.

Sticky Policies

This pattern implements the Design Strategy **Enforce**.

The goal of the pattern is to enable users to allow users to control access to their personal information.

For example, a service provider Service providers can use an obligation management system. Obligation management handles information lifecycle management based on individual preferences and organizational policies. The



obligation management system manipulates data over time, ensuring data minimization, deletion, and notifications to data subjects.

Federated Privacy Assessment

This pattern implements the Design Strategy **Demonstrate**.

We are in the context of Identity Management scenarios (that is, when the roles of the Identity Provider and the Service Provider are separated).

Identity Management solutions were introduced to decouple the functions related to authentication, authorization, and management of user attributes, on the one hand, and service provision on the other hand. Federated Identity Management allows storing a data subject's identity across different systems. All together, these form a Federation that involves complex data flows.

Federated Management solutions can be used to improve privacy (e.g., by allowing service providers to offer their services without knowing the identity of their users). However, the complexity of data flows and the possibility of collusion between different parties entail new risks and threats regarding personal data.

3.1.4 Privacy Report

Principle of Privacy by Design	Vulnerabilities									
	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
Proactive not reactive	x	-	-	-	x	-	x	-	-	-
Privacy as Default	x	-	x	-	-	-	x	-	-	-
Privacy embedded into Design	x	-	x	-	-	-	x	-	-	-
Full Functionality	-	-	-	-	x	-	x	-	-	-
End-to-End security	x	-	x	-	-	-	x	-	-	-
Visibility and Transparency	x	-	x	-	-	-	x	-	-	-
Respect for User Privacy	x	-	x	-	-	-	x	-	-	-

Principle of Privacy by Design	Privacy Design Strategies							
	Minimize	Hide	Separate	Abstract	Inform	Control	Enforce	Demonstrate
Proactive not reactive	x	x	x	x	-	-	-	-
Privacy as Default	x	x	x	x	-	-	-	-
Privacy embedded into Design	x	x	x	x	x	x	-	-
Full Functionality	x	x	x	x	x	x	-	-
End-to-End security	x	x	x	x	-	-	-	-
Visibility and Transparency	-	-	-	-	x	-	x	x
Respect for User Privacy	-	-	-	-	x	x	-	-

3.1.5 Design Strategies

In the diagrams that have been designed, the architecture is a Client-Server. The main difference respect to the original system is that the latter has only a server side which includes all the logic of Civic Sense project while the re-engineered one has a client side (in addition to the server one).

An example of Client can be a mobile or desktop application which communicates with the server sending and receiving data.

There have been identified three actors that can interact with the system:

1. **Admin:** the system administrator is the one which assigns the reports (which are notified by the User) to the Team. It is the main responsible of the entire system.
2. **Team:** it is the entity which take charge of the reports on the base of the instructions given by the Admin. It is able to login to the system and display all the reports which are assigned to it and eventually change the status (e.g., from “pending” to “on working” or from “on working” to “completed”).
3. **User:** is the last entity involved in the system, the one which notifies on the system the reports regarding problems of different kinds:
 - a. street lightning;
 - b. green areas;
 - c. waste and street cleaning;
 - d. streets and sidewalks;
 - e. signage and traffic lights.

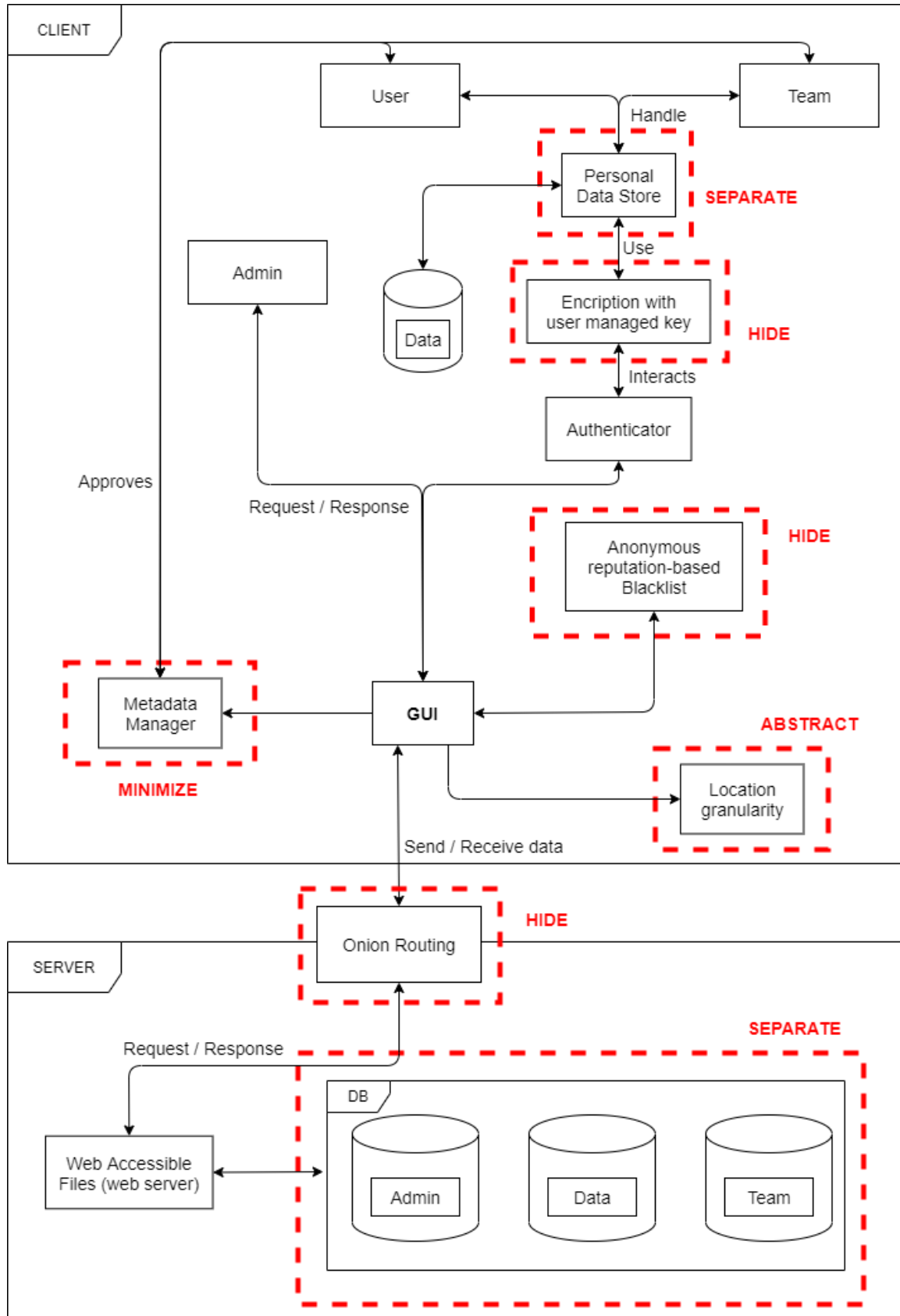
A use case of Civic Sense (with the actors explained above) can be the one that involves the city municipality context. In this case, there would be the server inside the building with the administrators which are the personnel that works inside it and have a direct access to the system. They also manage the Teams and the Users which are seen as external entities but however actors of the system.

The team can be thought as a company which is skilled in a specific area (like electricians or dustmen). It receives from the municipal employee a report which refers to a problem notified by a User and try to solve it.

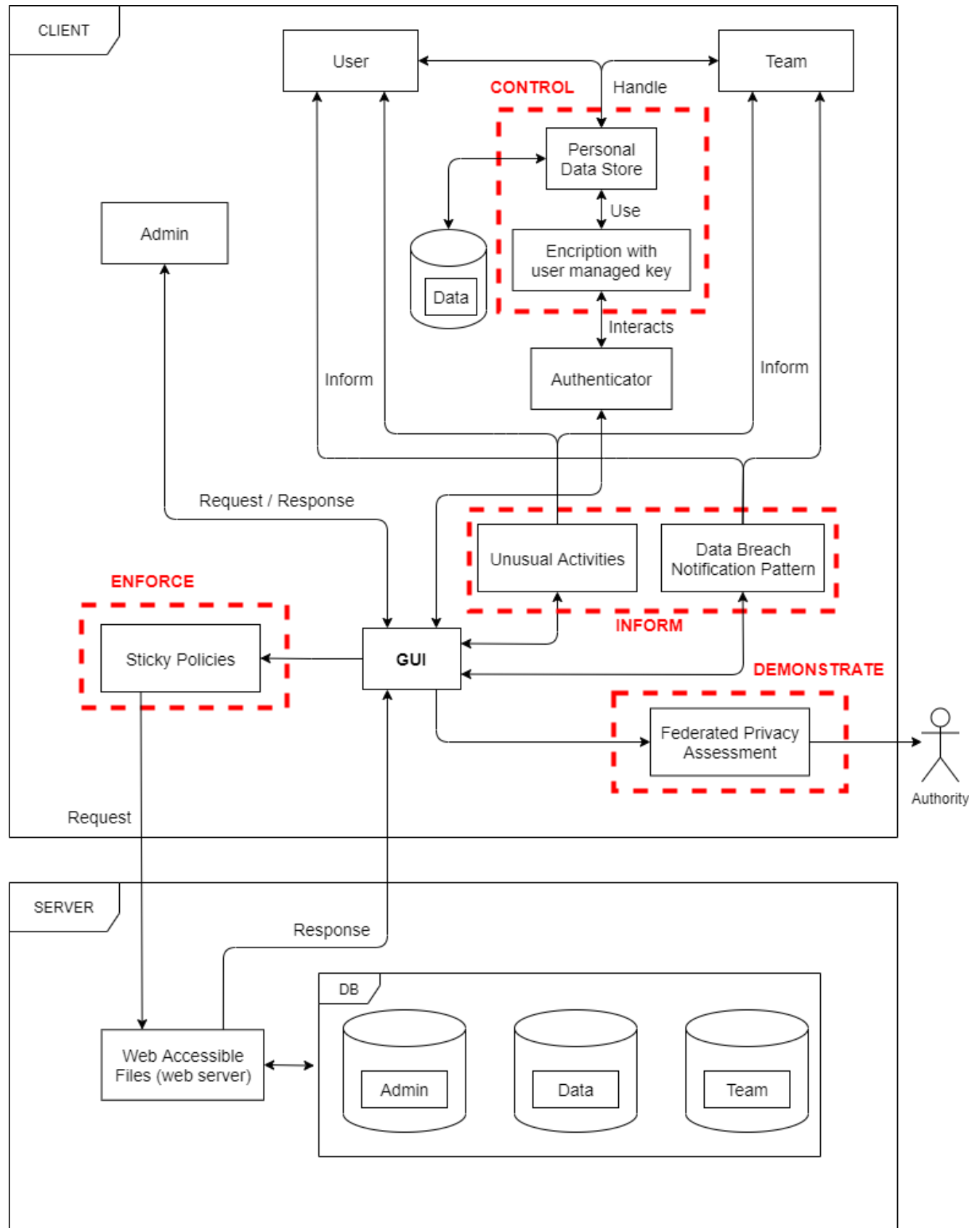
On the other way the User, as just said, is the responsible for reporting a problem concerning one of the five fields described above. In this use case the User can be a citizen who takes a photo of the problem identified and shares it in the report with its position (latitude, longitude), his email address

and other information automatically gathered like the data or the time of the reporting.

Data Oriented Design Strategy



Process Oriented Design Strategy



Description of the architecture diagrams

In the Data Oriented Design Strategy diagram, User and Team enhance the control on their personal data with the '**Personal Data Store**' pattern choosing which data they are going to share with the central server and which data they are going to keep stored on their personal devices (illustrated in the image as a little database connected with the pattern).

The information that are going to be sent to the central server are encrypted using the '**Encryption with user-managed keys**' pattern. In this way the user generates a strong encryption key and manages itself, keeping it private and unknown to the untrusted online services.

The login phase is performed using an authenticator (like Google Authenticator, Microsoft Authenticator or Authy).

The authenticated user is able now to interact with the system by a GUI (Graphical User Interface) which provides all the functionalities of Civic Sense.

The first important pattern connected to the GUI is the '**Metadata Manager**' that is responsible for the shared metadata. With the user's approval, it allows users that are not aware of the attached metadata to manage them and then eventually delete them.

The '**Anonymous reputation-based Blacklist**' pattern is used to assign to every user responsible of a reporting, a reputation score. In this way, for example, the session of the user (IP, device and last operation) who misbehave is recorded and stored in a blacklist.

'**Location granularity**' pattern allows the users to choose which location data must be shared: the name of the city, the province, the region, the country, or the exact latitude and longitude.

The pattern which permits a secure and private connection to the server is the '**Onion Routing**'. It adds anonymity by performing an encapsulation of the data in different levels of encryption.

In the server side, there is the 'Web Accessible Files (web server)' module which communicates with the Onion Routing performing requests and receiving responses and on the other way it is connected to the main databases of the system Civic Sense. By storing in different databases, the data and the information regarding the reports, the administrators' and

teams' sensitive information, it's ensured the implementation of the Privacy Design Strategy 'Separate'.

As for the Data Oriented Privacy Strategy diagram, also in the Process Oriented Privacy Strategy diagram we have that User and Team enhance the control on their personal data with the '**Personal Data Store**' pattern but in this case, it implements the 'Control' Design Privacy Strategy together with the '**Encryption with user-managed keys**', while in the first case (Data Oriented) they were playing the role of 'Separate' and 'Hide' respectively.

From the last pattern mentioned, as in the Data Oriented case, there is a connection to an external Authenticator and then from this last one to the GUI.

From the GUI we have several Privacy Design patterns connected:

'**Unusual Activities**' with '**Data Breach Notification Pattern**' are grouped as 'Inform' Privacy Design Strategy. The first one identifies unauthorized accesses by means of cookies, metadata, browser, type of architecture, etc.

The second one, instead, informs and reacts when a data breach occurs. These two patterns complement each other and can work together to handle unauthorized access to personal data by informing the Team or the User (in this case).

'**Sticky Policies**', which implements the 'Enforce' Design Privacy Strategy, is the pattern through which we make requests to the server: it allows to better specify the use of personal data and their processing.

The last pattern used in the Process Oriented Design Strategy architecture diagram is '**Federated Privacy Assessment**' which implements the 'Demonstrate' Privacy Design Strategy. It requires an external authority which controls the data subject's identity and provides an impact assessment on the privacy of the user. The assessment is made by various members (the authority previously mentioned) in order to define shared policies and demonstrate adequacy.



3.2 Explaining unresolved vulnerabilities with Privacy Patterns

Since most of all these remaining vulnerabilities has been found on external libraries, has been decided to use privacy pattern to explain how them can be solved. For each category, a Privacy Pattern has been found.

3.2.1 Privacy Violation (Critical)

Privacy Violation has been categorized as A3 – Sensitive Data Exposure by the OWASP Top 10 2017, by the Fortify scan tool. The instances founded are 6 and they are all referred to the library ‘class.smtp.php’ of the library PHPMailer.

In this case, some metadata may need to remain unencrypted to support the 3rd-party functions, for example file names for cloud storage, or routing information for transfer applications, exposing the metadata to risks of unauthorized access, server-side indexing for searching, or de-duplication.

The privacy pattern identified and more compliant to this category is **Encryption with user-managed keys**.

The system should be re-engineered such that the users can protect their privacy, and specifically the confidentiality of their personal information.

The solution is to let the user to generate and manage a strong encryption key to keep his personal information private, avoiding the transferring through untrusted 3rd-party services (like the SMTP protocol implemented in the PHPMailer library).

3.2.2 JavaScript Hijacking: Vulnerable Framework (Low)

“JavaScript Hijacking: Vulnerable Framework” doesn’t appear in any of the ten categories of the OWASP Top 10 2017 (so it has been categorized “Not Set”). The instances found are 3 and are all referred to the jquery.data.Tables.js file contained in the JQuery library.

The privacy pattern identified and more compliant to this category is **Aggregation Gateway**.

It consists in encrypt, aggregate and decrypt at different places. Indeed, a service provider gets continuous measurements of a service attribute linked to a set of individual service users.

So, the purpose is to let the service provider have reliable access to the aggregated load at every moment, so as to fulfil its operating requirements, without letting it access the individual load required from each specific service user.

3.2.3 Cross-Site Request Forgery (Low)

“Cross-Site Request Forgery” doesn’t appear in any of the ten categories of the OWASP Top 10 2017 (so it has been categorized as “Not Set”). The instances found are 22: 1 in the login.php file, 2 in segnalazionisemafori.php file, 2 in segnalazioniverde.php file, 4 in team.php file, 2 in segnalazionirifiuti.php file, 2 in segnalazioniilluminazione.php file, 2 in segnalazionistrade.php, 1 in inserisci.php file, 3 in index.php file, 1 in segnalazionii.php, 1 in registrateam.php file and 1 in forgotpassword.html file.

The privacy pattern identified and more compliant to this category is **Strip Invisible Metadata** which consist in stripping potentially sensitive metadata that is not directly visible to the end user.

Indeed, when a service requires a user to import data from external sources (e.g., pictures, tweets, documents) different types of metadata may be transmitted. Users may not be aware of the metadata as it can be automatically generated or not directly visible. Services might be inadvertently responsible for exposing private metadata or going against users' expectations.

Users are not always fully aware of the various kinds of metadata attached to files and web resources they share with online services. Much of this data is automatically generated, or not directly visible to users during their interactions. This can create situations where, even though users share information explicitly with services, they may be surprised to find this data being revealed. In certain cases where the data is legally protected, the service could be held responsible for any leakage of sensitive information.

The solution is so to stripping all metadata that is not directly visible during upload time, or during the use of the service can help protect services from leaks and liabilities. Even in cases where the information is not legally protected, the service can protect themselves from surprising their users and thus alienating them.