Integrating Security and Privacy in Software Development

Maria Teresa Baldassarre[1], Vita Santa Barletta[1*], Danilo Caivano[1], Michele Scalera[1]

[1] University of Bari Aldo Moro
Department of Computer Science, Via Orabona 4 - 70125 Bari, Italy
{mariateresa.baldassarre,vita.barletta,danilo.caivano,
michele.scalera}@uniba.it

Abstract. As a consequence to factors such as progress made by the attackers, release of new technologies and use of increasingly complex systems, threats to applications security have been continuously evolving. Security of code and privacy of data must be implemented in both design and programming practice to face such scenarios. In such a context, this paper proposes a software development approach, Privacy Oriented Software Development (POSD), that complements traditional development processes by integrating the activities needed for addressing security and privacy management in software systems. The approach is based on 5 key elements (Privacy by Design, Privacy Design Strategies, Privacy Pattern, Vulnerabilities, Context). The approach can be applied in two directions forward and backward, for developing new software systems or re-engineering an existing one. This paper presents the POSD approach in the backward mode together with an application in the context of an industrial project. Results show that POSD is able to discover software vulnerabilities, identify the remediation patterns needed for addressing them in the source code and design the target architecture to be used for guiding privacy-oriented system reengineering.

Keywords: Privacy by Design; Security by Design; Secure Software Development; Secure Architecture; System Reengineering; Cybersecurity; Application Security.

1 Introduction

Nowadays software systems and services impact technological layers and different application domains [1]. The growing dimension and complexity of software increase the range of cyber-attacks, the risk of information exfiltration and data breach. In this context, *Security* and *Privacy* play a major role in preserving the confidentiality, integrity and availability of data.

The number of attacks on information systems has been growing constantly in recent years [2]. The aim is to steal information and data by exploiting the vulnerabilities within the code [3]. This implies the need to identify and understand (at least) the most common threats to software security, disseminate security best practices, and address the security problem from the early stages of software development.

Security should be a basic feature of software applications such as automatically enabling complex password building mechanisms rather than procedures for renewing passwords periodically. The lack of system security can compromise privacy and for this reason privacy emerges as a proactive, integrative and creative approach to strengthen security requirements, starting from the design of new systems and protection of information assets and data in case of existing ones.

It becomes necessary to consider and pursue privacy throughout all the software life cycle phases. Today we can identify several approaches that address security, however they seldom consider the data privacy side of the problem. The same can be said for the current privacy-oriented approaches, i.e. privacy and security are addressed separately by the existing approaches. The challenges that companies and developer communities need to face within this context are many, but to start implementing defenses operatively, three major issues have to be addressed: (i) Translate best practices for both, secure application development and data privacy, into operational guidelines that can be traced back to code structures and software architectures; (ii) Share security and privacy competences within the development team. Privacy and security require specific skills that developers, even talented ones, often do not have. Therefore, it is necessary to share and transfer knowledge effectively [4, 5]; (iii) Integrate new methodologies for data privacy and secure software development into existing business processes. This must be done without affecting the legacy processes that are often peculiar to each company and consolidated over time [6].

This paper extends our previous work [7] and coherently, the main contributions it makes are summarized as follows: (i) proposal of an approach, *Privacy Oriented Software Development* (POSD), that is able to operationally support software development by integrating privacy and security requirements. It works on existing systems as well as on systems to be developed; (ii) introduction of a *Privacy Knowledge Base* (PKB), a knowledge base that supports decision making in all phases of the software lifecycle. PKB formalizes the relationship between 5 key elements and creates a navigation guide among them: Principles of Privacy by Design [8], Privacy Design Strategies [9], Privacy Pattern [10], Vulnerabilities [11], Context; (iii) the possibility to integrate the approach within the legacy processes used by companies without revolutionizing the latter but strengthening the process of secure development.

The paper is organized as follows: section 2 discusses related works on the topic of privacy and security in software development. In Section 3 the *Privacy Knowledge Base* is presented; Section 4 describes the approach adopted for the privacy-oriented software development in backward mode; Section 5 describes an application to a real case that shows how to apply POSD in backward mode. Sections 6 and 7, illustrate respectively the discussion of the results, the limitations of the work and conclusions.

2 Related Work

The security of software systems is constantly threatened by the increasing number of attacks. The aim of an attack is to exploit the vulnerabilities within the system's resources such as channels, methods, and data items [12]. Vulnerability is intended as one or more weaknesses that can be accidentally triggered or intentionally exploited and result in a violation of desired system properties [13]. Currently there are more than 140,000 vulnerabilities recorded [2]. Therefore, software development requires security principles to maintain the confidentiality, integrity and availability of the applications and the need to train professionals with respect to this dimension [14]. The concepts of security and privacy in software development are strictly related to each other and in recent years have become of pressing relevance due to the effect of the GDPR [15].

A considerable effort has been made for integrating security principles in software development processes. Some researchers have proposed strategies and frameworks for integrating security practices within the software development life cycle (SDLC) [16], but whatever model is adopted for secure software development, there is still the need for improvements in terms of metrics [17], penetration testing, training in secure development [18], as well as the need to practically integrate security policies for data into software application transactions during the development phases [19]. For example, Hilbrich et al. [20] propose a strategy for a clear and engineer like decision making process and to include security and privacy requirements in software development processes. The model definition specifies what has to be documented and how it has to be documented to avoid misunderstandings, support reproducibility, analysis and formal controls. Furthermore, Yanbing et. al [21] propose a Framework of a Software-Defined Security Architecture (SDSA) which can effectively decouple security executions with security controls, reduce the cost of software developments, and enhance the scalability of systems. In [22] an integrated security testing framework for Secure Software Development Life Cycle (SSDLC) was proposed to transform activities of SDLC into physical and executable test cases and thus to minimize the vulnerabilities. To quantitatively evaluate the security dimensions during the software production phase and enhance the overall security, Farhan et al. [23] add further steps to SDLC, i.e. follow the organization process, apply peer review, take care of testing and tracking the measure of security on SDLC. However, despite considerable efforts in this direction, many systems are being compromised and vulnerabilities are increasing. As so, the gap between the strategies and frameworks proposed and their actual application is impacted by the growth of attacks.

With respect to privacy, Privacy by Design (PbD) [8] is an approach to address data protection during software development and to integrate privacy throughout the system development lifecycle. The key problem in this approach is the lack of guidelines on how to map legal data protection requirements into system requirements and components. Privacy Design Strategies [9] represent an attempt to reduce the gap between "What to do" and "How to do it". Moreover, in [24] the authors try to correlate and map the available strategies with the "Privacy Patterns" needed to implement them, but the results obtained are limited in scope and far from being used in practice.

Privacy Patterns [25] represent a general solution to the most frequent privacy problems in software development. A further step in this direction was made by Suphakul et al. [26] where the proposed design patterns include information about privacy principles addressed and relevant software models in the UML notations to use [27]. In [28] authors propose a set of privacy process patterns for creating a clear alignment between privacy requirements and Privacy Enhancing Technologies (PET) [29], and encapsulate expert knowledge of PET implementation at the operational level.

Privacy must be integrated into the design to have strong security [30]. Six protection goals are analyzed in [31] and a common scheme for addressing the legal, technical, economic and social dimension of privacy and data protection in complex IT systems is provided. PRIPARE (Preparing Industry to Privacy by Design by supporting its Application in Research) [32] begins to highlight how privacy requirements can be incorporated into the SDLC. The study introduces a systematic methodology for privacy engineering, while Privacy-Friendly Systems Design [33] incorporate privacy through steps: elicitation of privacy requirements; analysis of the impact on the process; identification of supporting techniques. In these methodologies, and also in PriS [34], privacy is addressed during construction or early design activities instead of in all the phases.

Privacy by Design in itself lacks concrete tools to help software developers design and implement privacy friendly systems. It also lacks clear guidelines regarding how to map specific legal data protection requirements on the system [35]. In today's environment, privacy needs to be integrated into software development to protect sensitive data in growing systems and to enhance software quality. The principle of Full Functionality of the Privacy by Design [8] underlines this need, as well as the need to integrate the privacy and security dimensions.

However, most of the published literature deals with a single dimension, integrating either privacy elements or security elements into software development. There are obvious weaknesses in these approaches as they: are unable to implement solutions that are applicable in real contexts as they remain general in the definition and are far from being operative; focus their attention on software systems under development and do not address existing ones; represent new approaches to software development and can be adopted in place of those already in use.

The approach presented in this work (POSD) strives to overcome these weaknesses: (i) POSD integrates privacy and security practices by providing guidelines to developers that can be translated into operational practices, software architecture and software code. (ii) POSD can be used both when developing a software system and reengineering an existing one. (iii) POSD can be used jointly with the legacy development processes adopted.

3 Privacy Knowledge Base

The proposed approach, named Privacy Oriented Software Development (POSD), provides operational guidelines for integrating security and privacy management into any development process. It uses a *Privacy Knowledge Base* (PKB) to support decision making in system development and reengineering. PKB is based on the following 5 Key Elements (Fig. 1): Principles of Privacy by Design, Privacy Design Strategies, Privacy Patterns, Vulnerabilities and Context.

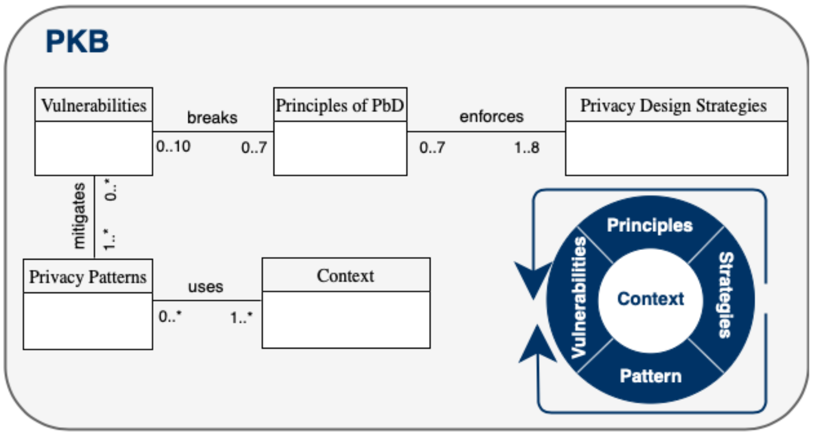


Fig. 1. The relationship between the Key Elements in PKB

The PKB defines which Principles of the Privacy by Design are violated by a specific vulnerability and which Privacy Design Strategies must be adopted to mitigate it. Operatively, given a selected strategy, a set of Privacy Patterns are associated to it so that the privacy requirements can be implemented. PKB has been developed following a Model-View-Controller architecture: the model provides all the methods to access the elements of the PKB; the view visualizes the relations between the 5 elements and manages interaction with the developer; the controller receives user requests and fulfills them changing the status of the two components.

Furthermore, the PKB engine integrates the results of static code analysis so that each vulnerability identified in the legacy system is associated to a privacy pattern and the architectural defects are fixed. Thanks to a connector, the PKB can analyze the vulnerabilities identified by various static code analysis tools. Each identified pattern can then be exported, by means of the Translator, in a specific language (Fig. 2).

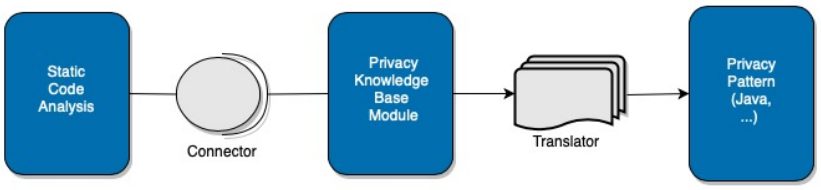
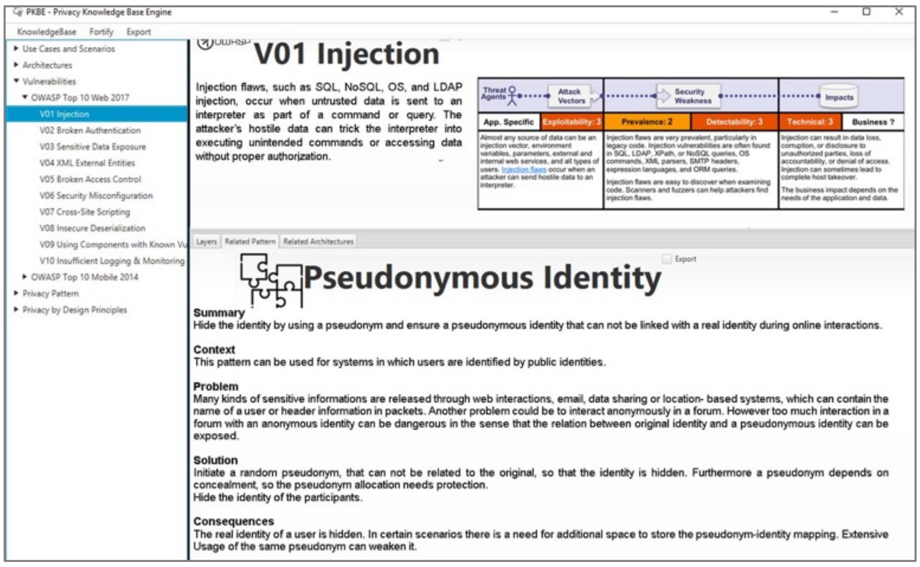


Fig. 2. Privacy Knowledge Base Engine

Thus, PKB provides guidelines to developers at all stages of the software lifecycle. These guidelines can be translated into operation by providing the necessary elements for system architecture design and coding. PKB can also be used on existing systems and on systems to be developed. It can be seen as a guided navigation between the key elements starting from any of these (Fig. 3).


PKBE - Privacy Knowledge Base Engine
KnowledgeBase
Fortify
Export
Use Cases and Scenarios
Architectures
Vulnerabilities
OWASP Top 10 Web 2017
V01 Injection
V02 Broken Authentication
V03 Sensitive Data Exposure
V04 XML External Entities
V05 Broken Access Control
V06 Security Misconfiguration
V07 Cross-Site Scripting
V08 Insecure Deserialization
V09 Using Components with Known Vu
V10 Insufficient Logging & Monitoring
OWASP Top 10 Mobile 2014
Privacy Pattern
Privacy by Design Principles
OWASP
Threat Agents
Attack Vectors
Security Weakness
Impacts
Layers
Related Pattern
Related Architectures
Export

V01 Injection

Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.

App. Specific	Exploitability: 3	Prevalence: 2	Detectability: 3	Technical: 3	Business ?
Almost any source of data can be an injection vector, environment variables, parameters, external and internal web services, and all types of users. Injection flaws occur when an attacker can send hostile data to an interpreter.		Injection flaws are very prevalent, particularly in legacy code. Injection vulnerabilities are often found in SQL, LDAP, XPath, or NoSQL queries, OS commands, XML parsers, SMTP headers, expression languages, and ORM queries. Injection flaws are easy to discover when examining code. Scanners and fuzzers can help attackers find injection flaws.		Injection can result in data loss, corruption, or disclosure to unauthorized parties, loss of accountability, or denial of access. Injection can sometimes lead to complete host takeover. The business impact depends on the needs of the application and data.	

Pseudonymous Identity

Summary
Hide the identity by using a pseudonym and ensure a pseudonymous identity that can not be linked with a real identity during online interactions.

Context
This pattern can be used for systems in which users are identified by public identities.

Problem
Many kinds of sensitive informations are released through web interactions, email, data sharing or location- based systems, which can contain the name of a user or header information in packets. Another problem could be to interact anonymously in a forum. However too much interaction in a forum with an anonymous identity can be dangerous in the sense that the relation between original identity and a pseudonymous identity can be exposed.

Solution
Initiate a random pseudonym, that can not be related to the original, so that the identity is hidden. Furthermore a pseudonym depends on concealment, so the pseudonym allocation needs protection.
Hide the identity of the participants.

Consequences
The real identity of a user is hidden. In certain scenarios there is a need for additional space to store the pseudonym-identity mapping. Extensive Usage of the same pseudonym can weaken it.

Fig. 3. Privacy Knowledge Base

In the next paragraphs the key elements of the PKB are illustrated.

3.1 Principles of Privacy by Design

Privacy by Design (PbD) integrates the principles of privacy and data protection within all stages of the software development process. PbD is based on seven principles, each of which specifies actions and responsibilities for evaluating "Privacy by Design Compliance" [8]:

- *Proactive not Reactive*: PbD anticipates and prevents privacy threats rather than reacting to privacy breaches once they have occurred.
- *Privacy as the default setting*: PbD seeks to deliver the maximum degree of privacy by ensuring that personal data are automatically protected in any given IT system or business practice. The privacy built into the system should not require any further user setup.
- *Privacy Embedded into Design*: Privacy becomes a core functionality being delivered. Therefore, it should not be implemented in response to a given event, but embedded in the design, IT system architecture and business logic.
- *Full Functionality*: PbD needs both, privacy and security. It aims to protect all the stakeholder's legitimate interests and objectives in a positive-sum "win-win"

perspective, thus avoiding that the interests of one party prevail over those of others by providing zero-sum solutions that imply an unnecessary trade-off.

- *End-to-End Security*: Strong security measures are essential to privacy, from start to finish. PbD ensures secure lifecycle management of information, end-to-end.
- *Visibility and Transparency*: Component parts and operations remain visible and transparent to users and providers. This ensures all stakeholders that whatever the business practice or technology involved, it will operate according to the agreed modalities and objectives.
- *Respect for User Privacy*: It requires architects and operators to treat the interests of the individual by offering specific solutions for a strong privacy default, appropriate notice and empowering user-friendly options.

3.2 Privacy Design Strategies

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 [9]. Eight privacy design strategies, based on the legal perspective of privacy, have been proposed. They are divided in two different categories [36]: *Data Oriented Strategies* and *Process Oriented Strategies.*

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 unlikable or 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.

3.3 Privacy Patterns

Privacy Patterns provide the knowledge collected from experts in a structured, documented and reusable manner [36, 38] and they contribute to a build secure information system. The solutions offered for using these patterns involve: detailing the information assets and the level of criticality of these assets; including the deployment details in a real environment, bearing in mind the architecture and the technologies that should be used; carrying out a qualitative analysis of the most important technological aspects with regard to the proposed solution [39]. Based on these considerations, privacy patterns that integrate privacy and security mechanisms [10, 25] were included in the PKB. This need was also expressed in [9] which highlights the importance of the patterns during the design phase. Therefore, privacy patterns support documenting common solutions to privacy problems; can improve the re-engineering of existing systems, describing classes, collaborations between objects and their purposes; help designers identify and address privacy concerns. Each pattern is structured according to the following characteristics:

- *Name*: represents the problem addressed.
- *Context*: contains a generic description of the setting and specifies the conditions under which the privacy pattern should be applied.
- *Problem*: the situation which has led to the need of applying privacy mechanisms and obtain a solution.
- *Solution*: describes the solution based on the scenario and the problem being considered.
- *Diagram*: represents the behavior and the structure of the pattern.

So, in this context a privacy pattern represents an answer to the following questions: (i) Which privacy design strategies must be implemented? (ii) Which vulnerabilities are mitigated/eliminated with the privacy solution?

An example of a Privacy Pattern is the following:
- *Name*: Anonymous Reputation-Based Blacklisting
- *Context*: A service provider provides a service to users who access anonymously, and who may make bad use of the service
- *Problem*: Anonymity may foster misbehavior. A 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 black list 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.
- *Solution*: First, the service provider provides their users with credentials for anonymous authentication. Then, 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. Note that these reputation values can only be linked to a specific session, but not to a specific user (as they have authenticated anonymously).
  When the user returns and starts a new session at the service, the service provider challenges the user to prove in zero-knowledge that he is not linked to any of the offending sessions (those that have a negative reputation associated). Zero-knowledge proofs allow the user to prove this, without revealing their identity to

the service provider. Different, alternative proofs have been proposed, e.g. prove that the user is not linked to any of the sessions in a set of session IDs, prove that the last K sessions of the user have good reputation, etc.

- *Diagram*: An example of a diagram is represented in Fig. 4

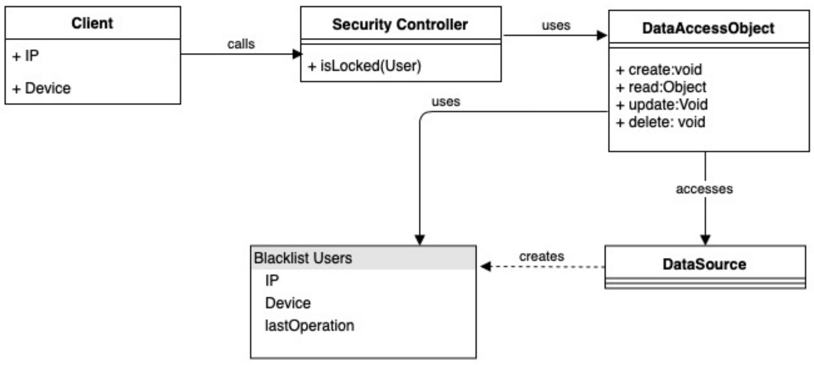


Fig. 4. Anonymous Reputation-Based Blacklisting

3.4 Vulnerabilities

The lack of sufficient logging mechanisms or not closing the database connection properly are some examples of vulnerabilities that allow an attacker to compromise software systems. A list of vulnerabilities classified according to the OWASP Top 10–2017 [9] has been integrated in the PKB. OWASP Top 10 is based primarily on data and information provided by firms specialized in application security or collected by using industry surveys. The goal of OWASP is to provide knowledge and information on the most common and important application security weaknesses. A short description of the macro categories of vulnerabilities included in the PKB follows:

A1-Injection: Untrusted data is sent to an interpreter as part of a command or query. For example, in SQL injection vulnerability, the root cause 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. The following code dynamically constructs and executes a SQL query that searches for items matching a specified name. The query restricts the items displayed to those where the owner matches the username of the currently authenticated user.

```
String userName = ctx.getAuthenticatedUserName();
String itemName = request.getParameter("itemName");
String query = "SELECT * FROM items WHERE owner = '"
     + userName + "' AND itemname = '"
     + itemName + "'";
ResultSet rs = stmt.execute(query);
```

The query that this code intends to execute follows:

```
SELECT * FROM items
WHERE owner = <userName>
AND itemname = <itemName>;
```

However, since the query is constructed dynamically by concatenating a constant base query string and a user input string, the query only behaves correctly if itemName does not contain a single-quote character. If an attacker with the user name wiley enters the string "name' OR 'a'='a" for itemName, then the query becomes the following:

```
SELECT * FROM items
WHERE owner = 'wiley'
AND itemname = 'name' OR 'a'='a';
```

The addition of the `OR 'a'='a'` condition causes the where clause to always evaluate to true, so the query becomes logically equivalent to the much simpler query:

```
SELECT * FROM items;
```

This simplification of the query allows the attacker to bypass the requirement that the query only return items owned by the authenticated user; the query now returns all entries stored in the items table, regardless of their specified owner.

A2-Broken Authentication: Authentication and session management not properly implemented allow attackers to compromise data and application. For example, a J2EE application can make use of multiple JVMs in order to improve application reliability and performance. In order to make the multiple JVMs appear as a single application to the end user, the J2EE container can replicate an `HttpSession` object across multiple JVMs so that if one JVM becomes unavailable another can step in and take its place without disrupting the flow of the application. In order for a session to be replicated, the values the application stores as attributes in the session must implement the `Serializable` interface.
The following class adds itself to the session, but because it is not serializable, the session can no longer be replicated.

```
public class DataGlob {
   String globName;
   String globValue;
   public void addToSession(HttpSession session) {
     session.setAttribute("glob", this);
   }
  }
```

A3-Sensitive Data Exposure: Sensitive data are not adequately protected in web applications and APIs and the attacker may steal or modify them to conduct credit card fraud, identity theft, or other crimes. For example, the following code contains a logging statement that tracks the contents of records added to a database by storing them in a log file. Among other stored values, we can find the return value from the `getPassword()` function that returns user-supplied plaintext password associated with the account.

```
<?php
  $pass = getPassword();
  trigger_error($id . ":" . $pass . ":" . $type . ":" .
  $tstamp);
?>
```

The code logs a plaintext password to the application eventlog. Although many developers trust the eventlog as a safe storage location for data, it should not be trusted implicitly, particularly when privacy is a concern.

A4-XML External Entities (XXE): Many XML processors evaluate external entity references within XML documents, so these entities can be used to disclose internal files. External entities can be used to disclose internal files using the URI handler, internal file shares, internal port scanning, remote code execution, and Denial of Service Attacks (DoS). It is crucial to use less complex data formats and avoid serialization of sensitive data; patch or upgrade all XML processors and libraries in use by the application or on the underlying operating system; disable XML external entity and DTD processing in all XML parsers in the application; implement positive server-side input validation, filtering, or sanitization to prevent hostile data within XML documents, headers, or nodes; etc. Consequently, to identify and mitigate this type of vulnerability, it is essential to train developers.
The following XML document shows an example of an XXE attack.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <!DOCTYPE foo [
  <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM file:///dev/random>]>
  <foo>&xxe;</foo>
```

This example could crash the server (on a UNIX system), if the XML parser attempts to substitute the entity with the contents of the /dev/random file.

A5-Broken Access Control: Attackers can exploit restrictions on authenticated users not properly enforced. This allows to access unauthorized functionality and/or data, modify or destroy data, or perform a business function outside of the user's limits. For example, 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.

```
id = Integer.decode(request.getParameter("invoiceID"));
String query = "SELECT * FROM invoices WHERE id = ?";
Query stmt = entmgr.createQuery(query).setParameter(0,id);
List invoices = stmt.getResultList();
```

The code uses a parameterized statement, which escapes metacharacters and prevents SQL injection vulnerabilities, to construct and execute a SQL query that searches for an invoice matching the specified identifier. The identifier is selected from a list of all invoices associated with the current authenticated user. On the other hand, the following example implements the same functionality but imposes an additional constraint requiring that the current authenticated user have specific access to the invoice.

```
userName = ctx.getAuthenticatedUserName();
id = Integer.decode(request.getParameter("invoiceID"));
String query = "SELECT * FROM invoices WHERE id = ? AND
user = ?";
Query stmt = entmgr.createQuery(query).setParameter(0,
id).setParameter(1, userName);
  List invoices = stmt.getResultList();
```

A6-Security Misconfiguration: Insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers and verbose error messages contain sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched and upgraded in a timely manner. The work of developers and system administrators is necessary to find misconfigurations and fix them such as the use of automated security scans and a periodic review of application, platform and server configuration guidelines. For example, processing XML documents can be computationally expensive. Attackers may take advantage of schemas that allow unbounded elements by supplying an application with a very large number of elements causing the application to exhaust system resources.
The following is an example of a schema that allows unbounded bar elements.

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
  <xs:element name="foo" >
    <xs:complexType>
      <xs:sequence>
  <xs:element name="bar" maxOccurs="unbounded" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  </xs:schema>
```

A7-Cross-Site Scripting (XSS): XSS occurs when an attacker executes scripts in the victim’s browser which can hijack user sessions, deface web sites, or redirect the user to malicious sites. The underlying flaws are that applications include untrusted data in a new web page without proper validation, updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. The following PHP code segment queries a database for an employee with a given ID and prints the corresponding employee's name.

```
<?php...
 $con = mysql_connect($server,$user,$password);
 ...
 $result = mysql_query("select * from emp where id="+eid);
 $row = mysql_fetch_array($result)
 echo 'Employee name: ', mysql_result($row,0,'name');
 ...
?>
```

This code functions correctly when the values of `name` are well-behaved, but it does nothing to prevent exploits if they are not.

It can appear less dangerous because the value of `name` is read from a database, whose contents are apparently managed by the application. However, if the value of `name` originates from user-supplied data, then the database can be a conduit for malicious content. Without proper input validation on all data stored in the data base, an attacker may execute malicious commands in the user's web browser.

A8-Insecure Deserialization: It often leads to remote code execution. Deserializations flaws can be used to perform attacks including replay attacks, injection attacks and privilege escalation attacks. An example is the following code that shows a PHP class implementing the `__destruct()` magic method and executing a system command defined as a class property. There is also an insecure call to `unserial-ize()`with user-supplied data.

```
class SomeAvailableClass {
    public $command=null;
    public function __destruct() {
            system($this->command);
    }
}
...
$user = unserialize($_GET['user']);
...
```

The application may be expecting a serialized `User` object but an attacker may actually provide a serialized version of `SomeAvailableClass` with a predefined value for its command property:

```
GET REQUEST:
http://server/page.php?user=O:18:"SomeAvailableClass":1
{s:7:"command";s:8:"uname -a";}
```

The destructor method will be called as soon as there are no other references to the `$user` object and then it will execute the command provided by the attacker. Attackers may chain different classes declared when the vulnerable `unserialize()` is being called using a technique known as “Property Oriented Programming”. This technique allows an attacker to reuse existing code to craft its own payload.

A9-Using Components with Known Vulnerabilities: Applications and APIs that used components with known vulnerabilities can facilitate an attack. Components, such as frameworks, libraries and other software modules, run with the same privileges as the application and if a vulnerable component is exploited, an attack can facilitate serious data loss or server takeover. An attacker identifies a weak component by scanning the system using automated tools or, less commonly, by analyzing the components manually. So, in software development it becomes necessary to identify all the components or libraries that applications use and the versions involved; establish security policies and best practices for component use; monitor known security vulnerabilities in any published databases, etc., and keep components upgraded to the latest available versions.
The following code uses a vulnerable method:

```
Double d = Double.parseDouble(request.getParameter("d"));
```

An attacker could send requests where the parameter `d` is a value in the range `[2^(-1022)- 2^(-1075) : 2^(-1022) - 2^(-1076)]`, such as "`0.0222507385850720119e-00306`", to cause the program to hang while processing the request. This vulnerability exists for Java version 6 Update 23 and earlier versions.

A10-Insufficient Logging & Monitoring: The breach is often caused by insufficient logging and monitoring, coupled with missing or ineffective integration with incident response. Attackers rely on the lack of monitoring and timely responses to achieve their goals without being detected. It is necessary to periodically test and validate computer systems, applications, related servers, and networks to rule out for example unlogged events, malicious activity alerts not detected in real time, alerts and subsequent responses that are not handled effectively, misconfigurations in firewalls and routing systems, locally stored logs without cloud backup.

3.5 Context

Data security implies that three fundamental characteristics are guaranteed: confidentiality, integrity, availability. Consequently, it is necessary to design the development scenario from beginning to end and identify the roles, the most significant scenarios, the technologies and the security mechanisms.

To ensure these objectives, a key element in the PKB is the context that integrates the necessary requirements to preserve the security of the data and the application. It consists of (i) Architectural Requirements; (ii) Use Cases and Scenario; (iii) Privacy Enhancing Technologies described below.

(i) *Architectural Requirements* determine the flow of data within the system, components, roles and responsibilities.
An example of how privacy patterns implement privacy design strategies within the selected architecture is given in Fig. 5**.**

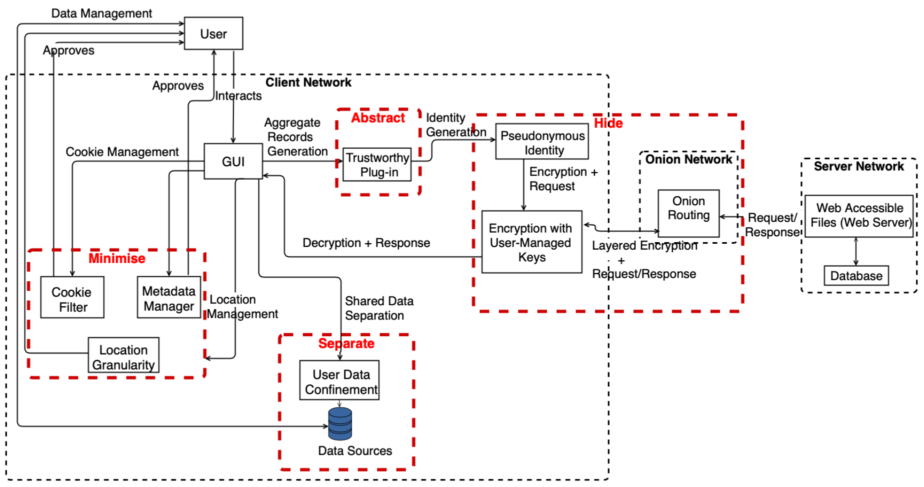


Fig. 5. Data Oriented Privacy Design Strategies in Client-Server Architecture

Fig. 5 illustrates the implementation of Data Oriented Strategies in a system with a Client-Server architecture. The operational flow is described below:

The user interacts with the system via the *Graphical User Interface* and entrusts the personal data management to the system. For principle of minimization, the system has three fundamental modules to reduce the personal data that can be shared on the network: *Metadata Manager* module acts when a user shares documents or web resources with third party services, it allows users who are not fully aware of the attached metadata, to manage them and then delete them; *Location Granularity* module to share user geographic location with a third party service. The system, through interface, and user approval, understands what granularity to use in sharing; *Cookie Filter* module avoids user monitoring.

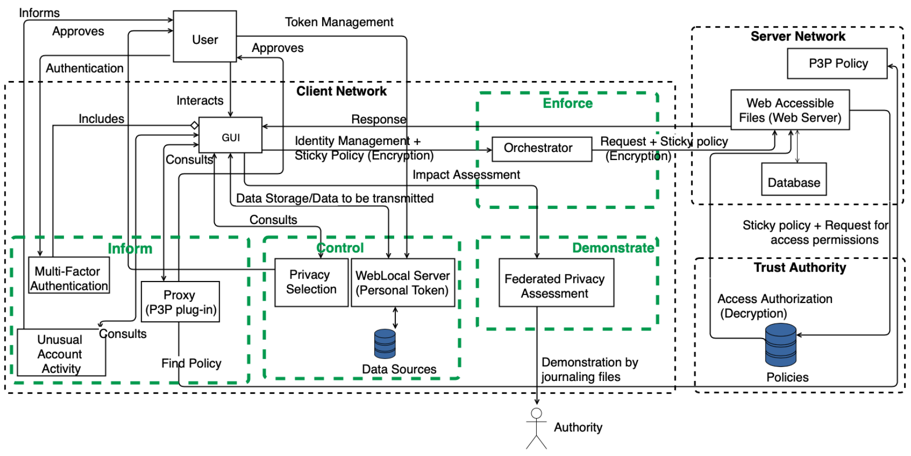
After minimizing the data, the system separates them so that the user's profile cannot be reconstructed: *User Data Confinement* module allows a user to manage personal data directly from his device. The data is separated so a third-party service cannot manage it.

The system then allows data to be aggregated by means of a module: *Trustworthy Plug-in* module creates a set of aggregated records, they are the set of all personal data grouped so that the server cannot correlate individual information with the user profile ensuring privacy.

Before sending the data to the server, the system hides it: *Encryption with user-managed keys* module enables encryption with keys managed directly by the user (asymmetric encryption). This hides all the aggregated records created; *Pseudonymous Identity* module generates a pseudonym identity to ensure anonymity in communication. The identity is generated before the client establishes a connection to the server so that no private information is known; *Onion Routing* module allows the identity to be hidden by encapsulating the data in different levels of encryption, limiting the knowledge of each node along the delivery path.

The Client makes requests to the Server.

The Server responds to the Client's request.



Informs
Approves
User
Token Management
Approves
Authentication
Server Network
P3P Policy
Client Network
Interacts
Includes
Response
Enforce
Web Accessible Files (Web Server)
GUI
Identity Management + Sticky Policy (Encryption)
Orchestrator
Request + Sticky policy (Encryption)
Consults
Impact Assessment
Database
Data Storage/Data to be transmitted
Consults
Sticky policy + Request for access permissions
Inform
Control
Demonstrate
Trust Authority
Multi-Factor Authentication
Privacy Selection
WebLocal Server (Personal Token)
Federated Privacy Assessment
Proxy (P3P plug-in)
Access Authorization (Decryption)
Consults
Unusual Account Activity
Find Policy
Data Sources
Demonstration by journaling files
Policies
Authority

Fig. 6. Process Oriented Privacy Design Strategies in Client-Server Architecture

On the other hand, Fig. 6 illustrates the use of process-oriented strategies in a client-server architecture:

The user interacts with the system via the *Graphical User Interface*.

The system provides three modules that are fundamental for the respect of the Inform principle: *Multi-Factor Authentication* module is integrated in the GUI and allows multi-factor authentication using CPU ID, UUID device and machine IP addresses; *Unusual Account Activity* module identifies unusual activities or unauthorized access by means of cookies, metadata, browser, type of architecture, etc...; *Proxy module (P3P)* defines a proxy capable of understanding the policies produced by the server and translating them into a format that the user can understand.

The system allows the user to control the data by means of two modules: *Privacy Selection* module avoids overly general privacy practices and provides the possibility of defining an adaptive privacy level for the contents shared with the controller; *Web Local Server* module defines a local web server that, by interfacing with a local database, allows the management of personal tokens to be shared over the network.

The system demonstrates the Control Authority the system’s security (Confidentiality, Integrity, Availability), using a fundamental module: *Federated Privacy Assessment* provides an impact assessment on the privacy of the user applying strategies oriented to processes and data on the system. The assessment is made by various members in order to define shared policies and demonstrate adequacy.

The system allows the enforcement of existing legal obligations by means of two modules: *Sticky Policies* are policies directly included in the requests made to the service provider. They allow to better specify the use of personal data and their processing; *Orchestrator* acts on behalf of the user and is controlled only by the latter, it ensures that the Identity Broker cannot correlate the original request from the service provider with the assertions that are returned by the Identity Provider.

The server responds to the client's request.

(ii) *Use cases and scenario* define all interactions with the system. The aim is to protect the information from unauthorized reading and manipulation.

Examples of use cases is given below, showing the threats that can determine it and the general countermeasures suggested.

Use Case: **User Registration/User Cancellation**
Threat: user abuse of privileges.
Countermeasures: define a process for granting and revoking an account that at least includes the use of individual User ID so that users can be made responsible for their own actions. The use of the group ID should be allowed only for: business or operational needs after approval and production of supporting documentation; verifying that the level of access requested is in line with the principle of “need to know”; immediately disabling or removing the User ID of users who no longer use the system (for example, the end of the employment relationship); periodically (at least quarterly) verifying the absence of inconsistent, redundant or obsolete accounts and their elimination.

Use Case: **Authentication**
Threat: unauthorized access to information; stealing of authentication credentials; unauthorized use of privileges.
Countermeasures: configure password quality control functions for access to systems, so that the composition meets the criteria of length, complexity and uniqueness necessary to have a high robustness. This means that the password must be composed of an increasing number of characters (at least 8) depending on the criticality of the information to be defended (e.g. 15 characters for administrative users); must contain characters in at least three of the following four categories: (i) uppercase letters of the Latin alphabet (from A to Z), (ii) lowercase letters of the Latin alphabet (from a to z), (iii) numbers to base 10 (from 0 to 9), (iv) special non-alphanumeric characters; should not refer to something that someone else can easily guess or obtain using information about the person (e.g. names, phone numbers, etc.).
In addition, the following rules must be observed: prohibit predefined account names and rename standard accounts such as, for example, the administrator account; do not show the password on the screen when it is entered and do not give indications on its length; the temporary password must be

changed at the first log-on; it must be forced for all users, and particularly for administrators, to change the password periodically; the password change procedure must prevent the re-use of all passwords previously used and include an effective procedure that takes into account input errors; limit the number of attempts allowed in a given time period or, alternatively, block the account for end-user access after a given number of attempts; do not allow access until the log-on process has been successfully completed; validate the log-on information only upon completion of all input data; limit the time within which the log-on procedure must be completed; consider viewing the following information successfully after log-on: (i) date and time of the previous successful log-on, (ii) detail any failed log-on attempts since the last successful log-on; the persistence and transmission of passwords must be protected; contrast the possibility provided by the browser cache to allow access, implement a policy that allows the user to choose not to save credentials or to force this policy as default; track both successful accesses and failed access attempts; perform the Audit of unsuccessful accesses to detect attempts to hack passwords; always check and validate the source IP address of the client used by the user.

(iii) *Privacy Enhancing Technologies (PETs)* [29] are a set of tools and technologies that help to protect the personal information handled by the applications:

Anonymity System is an anonymization technique through which the User's sensitive data is made inaccessible and encrypted.
Privacy Preserving Authentication is a useful technology to hide sensitive data of the user.
Privacy preserving cryptographic protocols are widely used for secure application-level data transport.
Information retrieval is a technique that has been developed to ensure that the representation, storage, organization and access to objects containing information such as documents, web pages, online catalogs and multimedia objects are managed correctly.
Data Anonymization, that consists in making user data anonymous and impossible to recompose in order to go back to personal information.
Pseudonymity Systems, where the user is identified by a pseudonym and not by his own name.
Encryption techniques describe the set of techniques that apply to ensure that personal data belonging to a user is stored and made secure against possible external attacks.
Access control requires to define measures and procedures to prevent unauthorized access to locations and information systems where personal data is held.
Policy and feedback tools to inform the user of the privacy policies and how the data will be protected while using the service.

4 Privacy Oriented Software Development

The Privacy Oriented Software Development (POSD) approach is inspired by the Software Development Life Cycle framework presented in [16]. It is based on existing systems as well as on systems to be developed, allowing to integrate privacy and security elements. This thanks to PKB that identifies the key elements of the two dimensions and the relationships between them.

In order to overcome the weaknesses identified in section 2 and provide operational guidelines, inputs, tools and techniques, outputs have been identified at each stage of the approach. POSD provides the development team all the elements and guidelines needed to develop or re-engineer a software system in a secure and privacy-oriented manner. It can be used in two ways: *Forward* for developing new systems and *Backward* for reengineering existing ones. In this research work the backward mode is presented and then applied in an industrial case study.

4.1 POSD in Backward Mode

In this section all the phases of the POSD are briefly presented together with their inputs and outputs (Fig. 7).

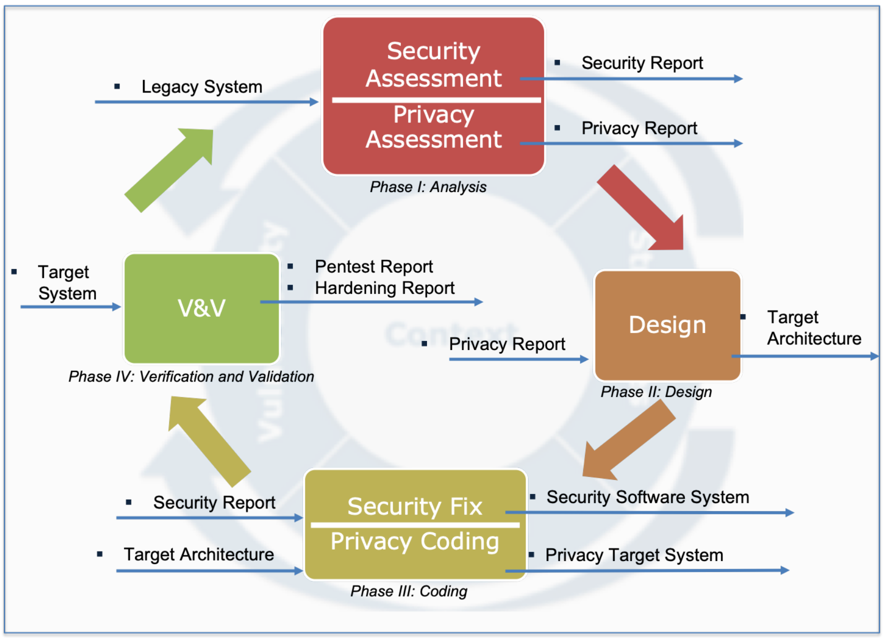


Fig. 7. Privacy Oriented Software Development (Backward Mode)

Phase I: Analysis. The analysis phase is divided into two parts: *Security Assessment* and *Privacy Assessment*. This derives from the need to analyze the system from both point of views, security and privacy. The *Security Assessment* consists in carrying out a static code analysis of the system to be re-engineered. The output is the *Security Report* containing the list of vulnerabilities in the system and a list of *Recommendation Patterns* for each category of vulnerability identified. An example is given below for the “Cross-Site Scripting: Persistent” vulnerability category:

- *Context*: The method *processGet()* in *JrInvocazioneWS.java* sends unvalidated data to a web browser on line 72, which can result in the browser executing malicious code.
  Problem: 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. In this case the data enters at *getEntity()* in *CheckMailPec.java* at line *95*.

  The data is included in dynamic content that is sent to a web user without being validated. The malicious content sent to the web browser often takes the form of a JavaScript segment, but may also include HTML, Flash or any other type of code that the browser may execute. 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.

  Example: The following JSP code segment queries a database for an employee with a given ID and prints the corresponding employee's name.

```
<%...
  Statement stmt = conn.createStatement();
  ResultSet rs = stmt.executeQuery("select * from emp
  where id="+eid);
  if (rs != null) {
     rs.next();
     String name = rs.getString("name");
  }
%>
Employee Name: <%= name %>
```

  This code functions correctly when the values of name are well-behaved, but it does nothing to prevent exploits if they are not. This code can appear less dangerous because the value of name is read from a database, whose contents are apparently managed by the application. However, if the value of name originates from user-supplied data, then the database can be a conduit for malicious content. Without proper input validation on all data stored in the database, an attacker may execute malicious commands in the user's web browser. This type of exploit, known as Persistent (or Stored) XSS, is particularly insidious because

the indirection caused by the data store makes it more difficult to identify the threat and increases the possibility that the attack will affect multiple users. XSS got its start in this form with web sites that offered a "guestbook" to visitors. Attackers would include JavaScript in their guestbook entries, and all subsequent visitors to the guestbook page would execute the malicious code.
Diagram: we provide an example of the sequence diagram (Fig. 8) for the vulnerability found in the *processGet()* method in *JrInvocazioneWS.java*

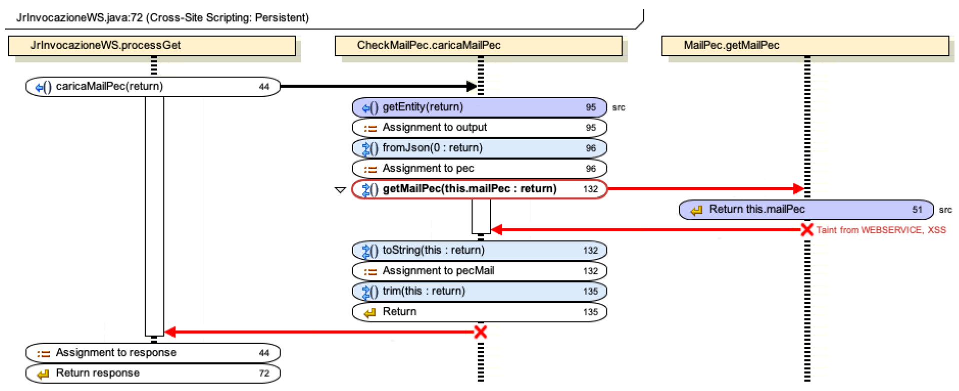

JrInvocazioneWS.java:72 (Cross-Site Scripting: Persistent)
JrInvocazioneWS.processGet
CheckMailPec.caricaMailPec
MailPec.getMailPec
caricaMailPec(return) 44
getEntity(return) 95 src
Assignment to output 95
fromJson(0 : return) 96
Assignment to pec 96
getMailPec(this.mailPec : return) 132
Return this.mailPec 51 src
Taint from WEBSERVICE, XSS
toString(this : return) 132
Assignment to pecMail 132
trim(this : return) 135
Return 135
Assignment to response 44
Return response 72

Fig. 8. Example of Cross-site scripting (XSS) vulnerabilities

- *Recommendation*: The solution to XSS is to ensure that validation occurs in the correct places and checks for the correct properties.
  Since 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.
  Web applications must validate their input to prevent other vulnerabilities, such as SQL injection, so augmenting an application's existing input validation mechanism to include checks for XSS is generally relatively easy. Despite its value, input validation for XSS does not take the place of rigorous output validation. An application may accept input through a shared data store or other trusted source, and that data store may accept input from a source that does not perform adequate input validation. Therefore, the application cannot implicitly rely on the safety of this or any other data. This means that the best way to prevent XSS vulnerabilities is to validate everything that enters the application and leaves the application designated for the user.
  The most secure approach to validation for XSS is to create a whitelist of safe characters that are allowed to appear in HTTP content and accept input composed exclusively of characters in the approved set. For example, a valid username might only include alpha-numeric characters, or a phone number might only include digits 0-9. However, this solution is often infeasible in web applications because many characters that have special meaning to the browser

should still be considered valid input once they are encoded, such as a web design bulletin board that must accept HTML fragments from its users.
A more flexible, but less secure approach is known as blacklisting, which selectively rejects or escapes potentially dangerous characters before using the input. In order to form such a list, you first need to understand the set of characters that hold special meaning for web browsers. Although the HTML standard defines what characters have special meaning, many web browsers try to correct common mistakes in HTML and may treat other characters as special in certain contexts, which is why we do not encourage the use of blacklists as a means to prevent XSS.
Details about special characters in various contexts are listed below:
In the content of a block-level element (in the middle of a paragraph of text):

- "<" is special because it introduces a tag.
- "&" is special because it introduces a character entity.
- ">" is special because some browsers treat it as special, on the assumption that the author of the page intended to include an opening "<", but omitted it in error.

The following principles apply to attribute values:

- In attribute values enclosed with double quotes, the double quotes are special because they mark the end of the attribute value.
- In attribute values enclosed with single quote, the single quotes are special because they mark the end of the attribute value.
- In attribute values without any quotes, white-space characters, such as space and tab, are special.
- "&" is special when used with certain attributes, because it introduces a character entity.

In URLs, for example, a search engine might provide a link within the results page that the user can click to re-run the search. This can be implemented by encoding the search query inside the URL, which introduces additional special characters:

- Space, tab, and new line are special because they mark the end of the URL.
- "&" is special because it either introduces a character entity or separates CGI parameters.
- Non-ASCII characters (that is, everything above 128 in the ISO-8859-1 encoding) are not allowed in URLs, so they are considered to be special in this context.
- The "%" symbol must be filtered from input anywhere parameters encoded with HTTP escape sequences are decoded by server-side code. For example, "%" must be filtered if input such as "%68%65%6C%6C%6F" becomes "hello" when it appears on the web page in question.

Within the body of a `<SCRIPT> </SCRIPT>`:

- Semicolons, parentheses, curly braces, and new line characters should be filtered out in situations where text could be inserted directly into a pre-existing script tag.

Server-side scripts:

- Server-side scripts that convert any exclamation characters (!) in input to double-quote characters (") on output might require additional filtering.

In the *Privacy Assessment*, the vulnerabilities identified during the static code analysis are provided as input to the PKB in order to identify: the principles of Privacy by Design violated by vulnerabilities (Table 1); the Privacy Design Strategies to be implemented in the system to respect the principles of Privacy by Design (Table 2); the privacy patterns that substantiate the Privacy Design Strategies (Table 3 gives an example). The results of this analysis are reported in the *Privacy Report.*

Table 1. Principles of Privacy by Design violated

Principles of Privacy by Design	**Vulnerabilities**									
	A1	A2	A3	A4	A5	A6	A7	A8	A9	A10
1. Proactive not Reactive	x	x	x	x	x	x	x	x	x	-
2. Privacy as the Default	x	x	x	x	x	x	x	-	x	-
3. Privacy Embedded into Design	x	x	x	x	x	-	x	-	x	x
4. Full Functionality	x	x	x	x	x	-	x	x	x	x
5. End-to-End Security	x	x	x	x	x	x	x	x	x	x
6. Visibility and Transparency	-	x	x	-	x	-	x	-	x	x
7. Respect for User Privacy	-	x	x	-	x	x	-	x	x	x

Table 2. Privacy Design strategies to implement

Principles of Privacy by Design		**Privacy Design Strategies**							
		Minimise	Hide	Separate	Abstract	Inform	Control	Enforce	Demonstrate
1	Proactive not Reactive	x	x	x	x	-	-	-	-
2	Privacy as the Default	x	x	x	x	-	-	-	-
3	Privacy Embedded into Design	x	x	x	x	x	x	-	-
4	Full Functionality	x	x	x	x	x	x	-	-
5	End-to-End Security	x	x	x	x	-	-	-	-
6	Visibility and Transparency	-	-	-	-	x	-	x	x
7	Respect for User Privacy	-	-	-	-	x	x	-	-

Table 3. Privacy Pattern that implement Privacy Design Strategies

Privacy Pattern		Privacy Design Strategies							
		Minimise	Hide	Separate	Abstract	Inform	Control	Enforce	Demonstrate
1	Strip Invisible Metadata	x	-	-	-	-	-	-	-
2	Protection Against Tracking	x	-	-	-	-	-	-	-
3	Location Granularity	x	-	-	-	-	-	-	-
4	Encryption with User-Managed keys	-	x	-	-	-	-	-	-
5	Pseudonymous Identity	-	x	-	-	-	-	-	-
6	Onion Routing	-	x	-	-	-	-	-	-
7	User Data Confinement	-	-	x	-	-	-	-	-
8	Trustworthy Privacy Plug-in	-	-	-	x	-	-	-	-
9	Privacy-Aware Network Client (P3P)	-	-	-	-	x	-	-	-
10	Handling Unusual Account Activities	-	-	-	-	X	-	-	-
11	Personal Data Store	-	-	-	-	-	x	-	-

12	Discouraging Blanket Strategies	-	-	-	-	-	x	-	-
13	Identity Federation Do Not Track	-	-	-	-	-	-	x	-
14	Sticky Policies	-	-	-	-	-	-	x	-
15	Federated Privacy Impact Assessment	-	-	-	-	-	-	-	x
16	Added-noise obfuscation	-	x	-	-	-	-	-	-
17	Data Breach Notification	-	-	-	-	x	-	-	-
18	Enable/Disable Functions	-	-	-	-	-	x	-	-
19	Pattern Matching Alghoritm	-	-	-	-	-	-	x	-
20	Preventing XSS with Filtering Approach	-	-	-	-	-	-	x	-
21	Anonymous Reputation-Based Blaclisting	x	x	x	-	-	-	-	-

Phase II: Design. This phase involves the design of a *Target Architecture* in order to re-engineer the system from a privacy point of view. The input for this phase is the *Privacy Report,* which contains, as previously described, the principles of PbD, privacy design strategies and the list of privacy patterns to use. The relationship between these elements support the team in designing a *Secure Software Architecture*. The output of the phase is the *Target Architecture*, i.e. the result of the application of the guidelines included in the *Privacy Report* to the original system. The general strategy followed by POSD is to integrate these guidelines with the minimum impact on the legacy system architecture and by preserving the control logic of the original system. For this reason, all the Privacy Patterns identified by PKB are included in two architectural components (Fig. 9), Data-Oriented and Process-Oriented components, that operationally translate the process flow metaphor of the eight privacy design strategies [9].

Phase III: Coding. The coding phase, as for the *Analysis* phase, is also divided into two parts: *Security Fix and Privacy Coding*. Starting from the vulnerabilities and the list of remediation patterns contained in the *Security Report*, the Security Fix will provide the *Secure Software System* in output, where all the vulnerabilities identified have been removed. This reduces the threat of attacks to the system. Instead, the *Privacy Coding* activity, starting from the *Target Architecture* defined in the previous phase and by using the *Secure Software System* obtained, will provide the *Target System* in output.

Phase IV: Verification and Validation. In this phase, before deploying the Target System, a *Penetration Test* and a Hardening phase are carried out in order to respectively verify the security level of the overall system and verify the correct setting of the base platform. The output of the penetration test activity is the *Pentest Report*, while for the hardening activity the *Hardening Report* is produced. Hardening refers to the set of specific configuration operations of a given IT system (and its related components) that aim to minimize the impact of possible cyber attacks that exploit its vulnerability, thereby improving its overall security. Hardening phase makes use of the CIS Benchmark [40] that consists in best practices for secure system configuration. The Penetration Test follows the guidelines of the OWASP testing guide [41].

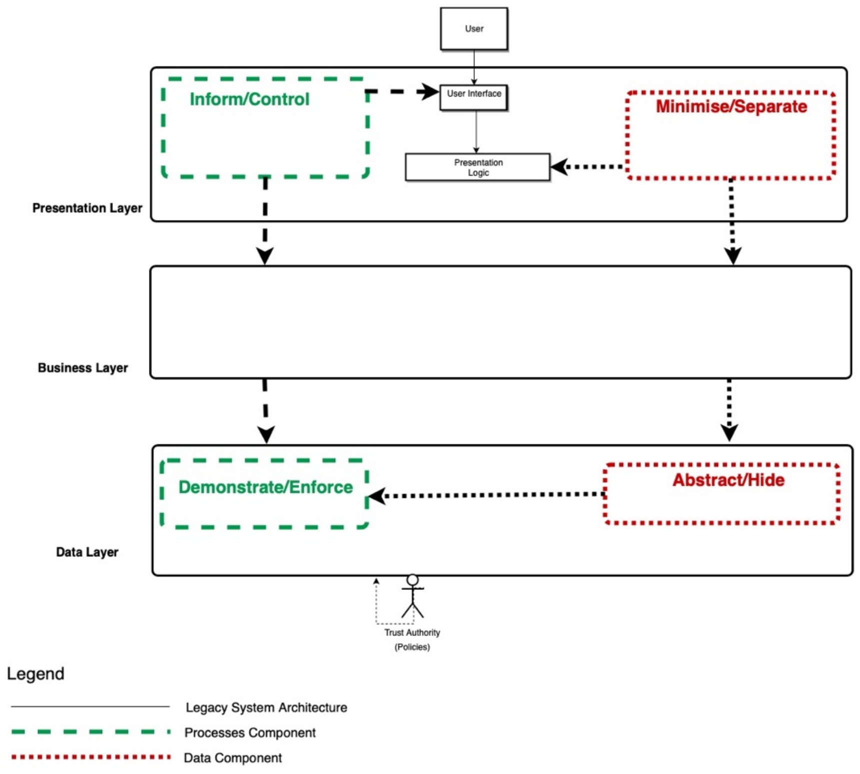


Fig. 9. Privacy Design Strategies in Target Architecture

5 Application to a Real Case

This section presents an ongoing industrial case study that shows the results of how POSD has been applied in backward mode for reengineering an existing legacy software system. The legacy system is used by a public company for processing the personal data of about one million users. The two main functionalities of the system are: acquisition and validation of the data of the subjects requesting the services of the public company and the verification of the economic financial reliability of the applicants. The legacy is a three tiers java system (Presentation, Business and Data). The re-engineering of the system involved a team of 5 people for a total of eight months (March 2019 - October 2019). The people involved in the project were 4 developers and 1 system administrator with more than 5 years of experience in the field but with no specific knowledge and competences on software security and privacy.

During the **Security Assessment** phase, static code analysis was carried out by using Fortify SCA [42] as source code analyzer. The project meta-information is reported as follows: (i) Number of Files: 371; (ii) Lines of Code: 125,105 (iii) Executable line of Code: 99,997; (iv) Total Vulnerabilities: 1318.

The number of vulnerabilities detected by the Fortify SCA analysis (Fig. 10) were further analyzed to exclude false positives. After removing false positives, the number of vulnerabilities was reduced to 1278.

Table 4 summarizes the number of remaining vulnerabilities classified according to the OWASP Top 10 2017 categories and severity, i.e. the probability that a vulnerability will be accurately identified and successfully exploited and the impact in terms of damage that an attacker could determine by successfully exploiting the vulnerability.

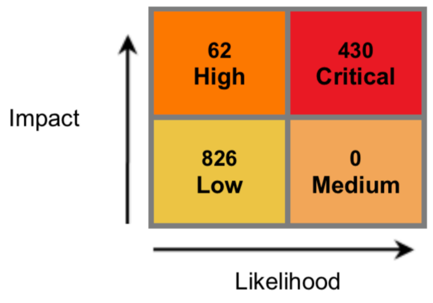


Fig. 10. Vulnerabilities by severity

The list of vulnerabilities produced by the *Security Assessment* activity were imported and analyzed in the PKB in order to identify the violated principles of PbD. The result showed that all 7 principles were violated, and thus the need to implement both data and process oriented strategies in the system by using the set of Privacy Patterns identified in the *Privacy Assessment* phase.

Table 4. Issues by OWASP Top 10 2017 Categories.

Vulnerabilities	Severity				Total
	Critical	High	Medium	Low	Issues
A1Injection	373	20	0	695	1088
A2 Broken Authentication	0	2	0	0	2
A3 Sensitive Data Exposure	21	3	0	1	25
A4 XML External Entities (XXE)	0	2	0	1	3
A5 Broken Access Control	0	33	0	87	120
A7 Cross-Site Scripting (XSS)	38	0	0	0	38
A9 Using Components with Know Vulnerabilities	0	0	0	2	2

The resulting list of Privacy Patterns to be applied for reengineering the system is shown in (Fig. 11) and the *Target Architecture* identified by the development team is available at [43]. For what concerns the *Coding* phase that includes *Security Fix* and *Privacy Coding*, the team had fixed 1200 vulnerabilities by applying the remediation patterns. The remaining 78 vulnerabilities refer to architectural flaws that affect privacy of the legacy system. They were removed by implementing the *Target Architecture*.

At the end of this phase, penetration test and hardening activities were carried out. The results obtained show that all the security and privacy issues that affected the original legacy system have been overcome.

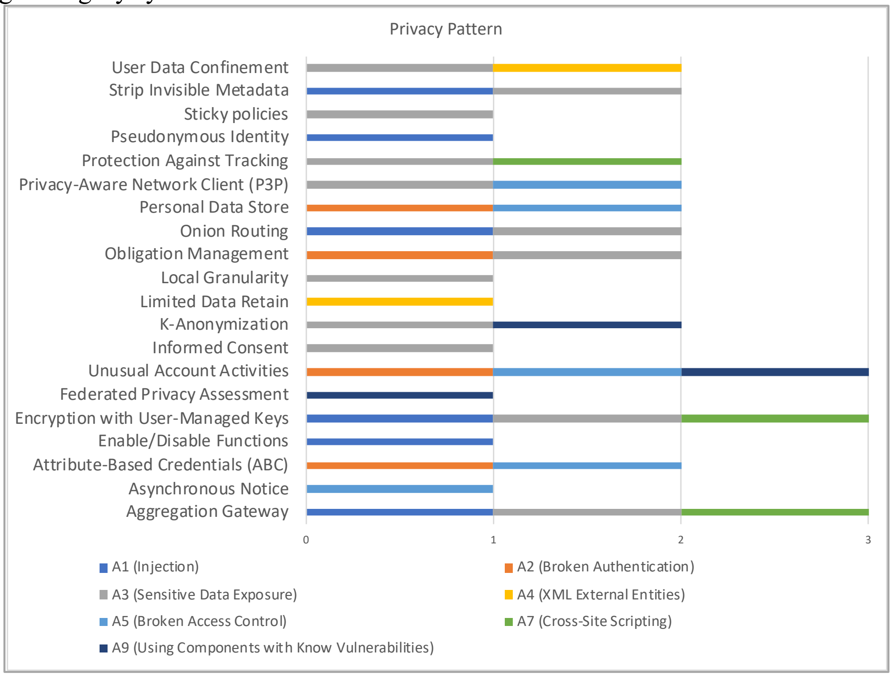


Fig. 11. Privacy Patterns applied vs class of vulnerabilities mitigated in the *Target System*

6 Discussion and limitations

Nowadays a large application park written in various languages and operating on diversified architectures and platforms is available. Its maintenance requires several and ample skills that are unlikely to all be found in a single developer.

Cyber Security is a relatively recent discipline that has received growing attention due to the impacts of the cyber-attacks received and of the damages (economical, image, etc.) produced. This is even more critical in the case of privacy. Interest in data privacy has increased in the past year due to the application of GDPR regulations that enforce a set of principles to protect users' privacy without providing operational guidelines on how to apply them.

In addition, it is also relevant to consider that developers are often not very sensitive to systemic problems underlying the operation of software systems and the provision of services. This latter field is important to software systems that, in turn, tend to neglect the problems related to development. It is no coincidence that in recent years development approaches such as DevOps, that attempts to bring the two worlds together, have emerged. Security and privacy impact both areas significantly.

The existing application park was for the most part designed and developed when issues related to cyber security and privacy were not perceived in such a way. Legacy systems have not been designed for privacy and security. Once again, we are called to fill the gap and remediate. This requires professionals with specific competences specifically trained for this purpose and, given the heterogeneity of the technologies and delivery platforms used, it is not easy to find them.

All these factors put together cause an objective difficulty to collect previous experiences and identify sufficient competences able to face security and privacy problems effectively.

In this scenario POSD and PKB represent a first attempt to fill the knowledge and competence gap assuring multi-level support, i.e. from analysis to design of architectures and coding, to the operational level. Furthermore, PKB provides knowledge and pre-configured solutions to known problems, whereas POSD provides operational guidelines.

Being integrated with a source code analyzer like Fortify SCA (but not limited to it) PKB is able to operate with 25 different programming languages among the most known and used along with the related *Remediation Patterns* included.

The *Privacy Patterns* identified in the PKB, on the other hand, represent architectural solutions that can be adopted beyond the languages and specific technologies used. The Hardening phase foreseen by the POSD uses multiple scripts (i.e. CIS Benchmarks) that are currently available for each operating system, application server, web server, mail server, browser and type of device, effectively guaranteeing an almost total coverage of the possible delivery platforms.

POSD and PKB represent a useful cookbook that can be used to address security and privacy problems on behalf of both developers and systems engineers, even in the absence of specific security and privacy skills.

The results obtained in the real case application of the POSD, show that it was able to address both, security and privacy issues. It allowed to fix the vulnerabilities identified during the static code analysis through the application of remediation patterns provided in the *Security Report* and furthermore to addresses the privacy flaws listed in the *Privacy Report*. The use of PKB across all the POSD phases has allowed to share knowledge and operative guidelines among the development team. The team was able to find vulnerabilities, understand the underlying problems and apply security fixes.

PKB supported the team in the privacy oriented reengineering of the legacy system, in assuming design decisions even though the team members had no specific skills and knowledge about security and privacy.

The use of POSD did not impact on the development process used within the organization. All the activities performed by the team, starting from the requirements provided by POSD, were carried out according to the software processes and procedures already used in the company without altering the modus operandi.

One of the limitations of this work is represented by the number and size of the software systems used for validation. As of now, only one software system has been reengineered; it was however, a real industrial system. Consequently, the authors are confident that the results obtained so far provide useful insights for addressing security and privacy issues.

7 Conclusions

This research work proposes an approach named Privacy Oriented Software Development (POSD) for addressing privacy and security problems in software systems.

It uses a Privacy Knowledge Base (PKB) that developers can use whenever necessary during software development or reengineering.

PKB is based on 5 interrelated key Elements for supporting decisions and choices in all phases of the software life cycle: Principles of Privacy Design, Privacy Design Strategies, Privacy Patterns, Vulnerabilities, Context.

The proposed approach can be applied forward, for developing a new system, and backward for reengineering an existing one.

In this paper the backward mode was presented and applied within an industrial case study. The results obtained show that it was able to provide best practices for both, secure application development and data privacy, operational guidelines, software architectures and code structures to use. This suggests that the proposed approach may be successfully used for addressing security and privacy problems also in absence of specific competences and skills, as in the case of the application case carried out and the team involved.

Despite the limitations of the validation carried out, this work has allowed us to start a discussion about this research idea and lay the foundations for future work such as: validate the POSD approach in forward mode on systems to develop; improve the PKB effectiveness by adding further knowledge (*Remediation Patterns* and *Privacy Patterns*) and increase the number of supported programming languages. At the moment the PKB is able to export and provide a Java based structure to users for almost all the privacy patterns it contains. In the future more programming language will be added in order to improve the efficacy of the PKB.

In this first work we preferred to start from a backward mode in consideration of the enormous amount of legacy systems currently existing on the market. They were developed in a time when neither security nor privacy were perceived as important as they are today, and thus represent an important commitment for software engineering scientific and industrial communities.

Acknowledgements

This study has been partially funded by the Project “Digital Service Ecosystem” (Cod. PON03PE_00136_1) funded by Italian Minister of University and Research and Project “Auriga2020” - (Cod. T5LXK18), funded by Apulia Region.

References

1. Baldassarre, M.T., Barletta, V.S., Caivano, D., (2018). Smart Program Management in a Smart City. *110th AEIT International Annual Conference, AEIT 2018*, https://doi.org/10.23919/AEIT.2018.8577379
2. IBM, X-Force Threat Intelligence Index 2019. Resource document. IBM Security. Accessed 17 October 2019.
3. Halkidis, S. T., Tsantalis, N., Chatzigeorgiou A. and Stephanides, G. (2008). Architectural Risk Analysis of Software Systems Based on Security Patterns. In *IEEE Transactions on Dependable and Secure Computing*, vol. 5, no. 3, pp. 129-142.
4. Baldassarre, M.T., Caivano, D., Visaggio, G. (2013). Empirical studies for innovation dissemination: Ten years of experience. In: *17th International Conference on Evaluation and Assessment in Software Engineering*, EASE 2013. ACM International Conference Proceedings Series, ACM Press, ISBN: 978-145031848-8, https://doi.org/10.1145/2460999.2461020
5. Ardimento, P., Caivano, D., Cimitile, M., Visaggio, G. (2008). Empirical Investigation of the Efficacy and Efficiency of Tools for Transferring Software Engineering Knowledge. *Journal of Information & Knowledge Management*, vol. 7 n.3, p. 197-207, ISSN: 0219-6492
6. Caivano, D., Fernandez-Ropero, M., Pérez-Castillo, R., Piattini, M., Scalera, M. (2018). Artifact-based vs. human-perceived understandability and modifiability of refactored business processes: An experiment. *Journal of Systems and Software*, Vol. 144, pp. 143-164.
7. Baldassarre M.T., Barletta V.S., Caivano D., Scalera M., (2019). Privacy Oriented Software Development. *Communications in Computer and Information Science*, 1010, pp. 18-32, https://doi.org/10.1007/978-3-030-29238-6_2
8. Cavoukian, A. (2012). Operationalizing Privacy by Design: A Guide to Implementing Strong Privacy Practices. Resource document. Global Privacy and Security by Design. http://www.ontla.on.ca/library/repository/mon/26012/320221.pdf. Accessed 17 October 2019.
9. Hoepman, J.-H. (2014). Privacy Design Strategies. In IFIP, ICT Systems Security and Privacy Protection (pp 446-459). Springer Berlin Heidelberg.
10. Privacy Patterns, https://privacypatterns.org. Resource document. UC Berkeley, School of Information. Accessed 17 October 2019.
11. OWASP Top 10 – 2017. The Ten Most Critical Web Application Security Risks. Resource document. OWASP. https://owasp.org. Accessed 17 October 2019.
12. Hatzivasilis, G., Papaefstathiou, I. and Manifavas, C. (2016). Software Security, Privacy, and Dependability: Metrics and Measurement. In *IEEE Software*,vol. 33, no. 4, pp. 46-54.
13. Black, P. E., Badger, L., Guttman, B., Elizabeth Fong, E. (2016). Dramatically Reducing Software Vulnerabilities. Resource document. National Institute of Standards and Technology (NIST). https://doi.org/10.6028/NIST.IR.8151 Accessed 17 October 2019
14. Baldassarre, M. T., Barletta, V. S., Caivano, D., Raguseo, D., Scalera, M. (2019). Teaching Cyber Security: The Hack-Space Integrated Model. Proceedings of the *Third Italian Conference on Cyber Security*, Vol-2315, CEUR Workshop Proceedings.
15. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC, (2016). Resource document. Official Journal of the European Union. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679. Accessed 18 October 2019.

16. Kissel, R. L., Stine, K. M., Scholl, M. A., Rossman, H., Fahlsing, J., Gulick, J. (2008). Security Considerations in the System Development Life Cycle. *Special Publication (NIST SP)* – 800-64 Rev 2.
17. Ardimento, P., Baldassarre, M.T., Caivano, D., Visaggio, G. (2004). Multiview framework for goal oriented measurement plan design. In *Lecture Notes in Computer Science* (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 3009, pp. 159-173., https://doi.org/10.1007/978-3-540-24659-6_12
18. Jaatun, M. G., Cruzes, D. S., Bernsmed, K., Tøndel, I. A., and Røstad, L. (2015). Software Security Maturity in Public Organisations. In International Conference on Information Security, *ISC 2015: Information Security* (pp 120-138). Springer International Publishing, Cham. https://doi.org/10.1007/978-3-319-23318-5_7
19. Navarro-Machuca, J., and Chen, L. (2016). Embedding Model-Based Security Policies in Software Development. *IEEE 2nd Int. Conf. on Big Data Security on Cloud (BigDataSecurity)*, IEEE Int. Conf. on High Performance and Smart Computing (HPSC), and IEEE Int. Conf. on Intelligent Data and Security (IDS), New York, NY, pp. 116-122.
20. Hilbrich, M., and Frank, M. (2017). Enforcing Security and Privacy via a Cooperation of Security Experts and Software Engineers: A Model-Based Vision. *IEEE 7th International Symposium on Cloud and Service Computing (SC2)*, Kanazawa, pp. 237-240.
21. Yanbing, L., Xingyu, L., Yi, J., and Yunpeng, X., (2016). SDSA: A framework of a software-defined security architecture. In *China Communications*, vol. 13, no. 2, pp. 178-188.
22. Tung, Y., Lo, S., Shih, J., and Lin, H. (2016). An integrated security testing framework for Secure Software Development Life Cycle. *18th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, Kanazawa, pp. 1-4.
23. Shehab Farhan, A. R., and Mostafa, G. M. (2018). A Methodology for Enhancing Software Security During Development Processes. *21st Saudi Computer Society National Computer Conference (NCC)*, Riyadh, pp. 1-6. https://doi.org/10.1109/NCG.2018.8593135
24. Colesky, M., Hoepman, J., and Hillen. C. (2016). A Critical Analysis of Privacy Design Strategies. *IEEE Security and Privacy Workshops (SPW),* San Jose, CA, pp. 33-40. https://doi.org/10.1109/SPW.2016.23
25. Thomborson, C. (2016). Privacy patterns. *14th Annual Conference on Privacy, Security and Trust (PST)*, Auckland, pp. 656-663.
26. Suphakul, T., and Senivongse, T. (2017). Development of privacy design patterns based on privacy principles and UML. *18th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)*, Kanazawa, pp. 369-375. https://doi.org/10.1109/SNPD.2017.8022748
27. Fernández-Sáez, A.M., Genero, M., Caivano, D., Chaudron, M.R.V. (2016). Does the level of detail of UML diagrams affect the maintainability of source code?: a family of experiments. *Empirical Software Engineering*, Volume 21, Issue 1, 1 February 2016, pp 212-259 https://doi.org/10.1007/s10664-014-9354-4
28. Diamantopoulou, V., Argyropoulos, N., Kalloniatis, C., and Gritzalis, S. (2017). Supporting the design of privacy-aware business processes via privacy process patterns. *11th International Conference on Research Challenges in Information Science*, Brighton, pp.187-198. https://doi.org/10.1109/RCIS.2017.7956536
29. Van Blarkom, G.W., Borking, J.J., and Olk, J.G.E. (2003). Handbook of Privacy and Privacy-Enhancing Technologies. The Case of Intelligent Software Agents. *College Bescherming Bersoonsgegevens*, ISBN 90-74087-33-7
30. Cavoukian, A. (2016). International Council on Global Privacy and Security, By Design. In *IEEE Potentials*, Sept.-Oct. 2016, vol. 35, no. 5, pp. 43-46.

31. Hansen, M., Jensen, M., and Rost, M. (2015). Protection Goals for Privacy Engineering. *IEEE Security and Privacy Workshops*, San Jose, CA, pp. 159-166 (2015).
32. Notario, N., Crespo, A., Martìn, Y. S., Del Alamo, J. M., Le Métayer, D., Antignac, T., Kung, A., et al. (2015). PRIPARE: Integrating Privacy Best Practices into a Privacy Engineering Methodology. *IEEE Security and Privacy Workshops*, San Jose, CA, pp. 151-158. https://doi.org/10.1109/SPW.2015.22
33. Spiekermann, S., and Cranor, L. F. (2009). Engineering privacy. *IEEE Transactions on Software Engineering*, vol. 35, no. 1, pp. 67–82. https://doi.org/10.1109/TSE.2008.88
34. Kallpniatis, C., Kavakli, E., Gritzalis, S. (2008). Addressing privacy requirements in system design: the PriS method. *Requirements Engineering*, vol. 13, pp 241-255. Springer-Verlag. https://doi.org/10.1007/s00766-008-0067-3
35. Morales-Trujillo, M. E., Matla-Cruz, E. O., García-Mireles, G. A., & Piattini, M. (2018). Privacy by design in software engineering: A systematic mapping study. Paper presented at *Avances en Ingenieria de Software a Nivel Iberoamericano*, CIbSE. pp.107-120.
36. Colesky, M., Hoepman, J., Hillen, C. (2016). A Critical Analysis of Privacy Design Strategies. *IEEE Security and Privacy Workshops (SPW)*, San Jose, CA, 2016, pp. 33-40. https://doi.org/10.1109/SPW.2016.23
37. Ortiz, R., Moral-Rubio, S., Garzás, J., Fernández-Medina. E. (2011). Towards a Pattern-Based Security Methodologiy to Build Secure Information Systems. Proceedings of the *8th International Workshop on Security in Information Systems WOSIS 2011*, pp. 59-69.
38. Baldassarre, M. T., Bianchi, A., Caivano, D., Visaggio, G. (2005). An Industrial Case Study on Reuse Oriented Development. In: Proceedings of *21st IEEE International Conference on Software Maintenance*. p. 283-292, WASHINGTON, DC: IEEE Computer Society, ISBN:0-7695-2368-4, Budapest Hungary, September 2005. https://doi.org/10.1109/ICSM.2005.20
39. Moral-García, S., Ortiz, R., Moral-Rubio, S., Vela, B., Garzás, J., Fernández-Medina, E. (2010). A New Pattern Template to Support the Design of Security Architectures. *PATTERNS 2010: The 2nd Int. Conferences on Pervasive Patterns and Applications*, pp. 66-71.
40. Center for Internet Security (2019). CIS Benchmarks. Resource document. CIS. https://www.cisecurity.org/cis-benchmarks/. Accessed 21 October 2019
41. OWASP Testing Guide, (2016). Resource document. OWASP. https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents. Accessed 22 October 2019
42. Fortify Static Code Analyze (SCA), (2019). Resource document. Micro Focus. https://www.microfocus.com, 2018. Accessed 22 October 2019
43. Baldassarre, M. T., Barletta, V. S., Caivano, D., Scalera, M. (2019). Target Architecture in Privacy Oriented Software Development. SERLAB. https://serlab.di.uniba.it/posd