Patient Community – A Test Bed For Privacy Threat Analysis

Immanuel Kunz, Angelika Schneider, Christian Banse, Konrad Weiss, Andreas Binder

Fraunhofer AISEC, Garching b. München, Germany

{firstname.lastname}@aisec.fraunhofer.de

Motivation

Research and development of privacy analysis tools currently suffers from a lack of test beds for evaluation and comparison of such tools.
In the area of security, analysis tools and respective benchmarks are well researched and maintained, while there is little research into privacy-related tooling and benchmarks. To the best of the authors' knowledge, an application with real deployment configurations as a privacy benchmark has not been proposed before.

What is our goal? We aim to provide a test bed that encourages researchers and practitioners to develop and test privacy analysis tools, use it for educational purposes, as well as a basis for discussion about the code- and deployment-level analysis of privacy weaknesses.

What exists today? For security testing and learning, there is the *Damn Vulnerable Web Applications Directory* by OWASP [1]. Also, there are test suites for testing analysis tools, like the Juliet test suite [2]. **However, a comparable approach for privacy is missing**.

LINDDUN: LINDDUN is a privacy threat modeling framework that uses the privacy threats Linkability, Identifiability, Non-repudiation, Detectability, Disclosure, Unawareness, and Policy non-compliance. In our test bed, we aim at implementing as many types of LINDDUN threats as possible. The most recent version is LINDDUN GO [4].

Architecture

The Patient Community Example (PCE) consists of multiple microservices which are structured as shown in Figure 1, and further explained in Table 1. It was first described in an example LINDDUN analysis by Wuyts [3].

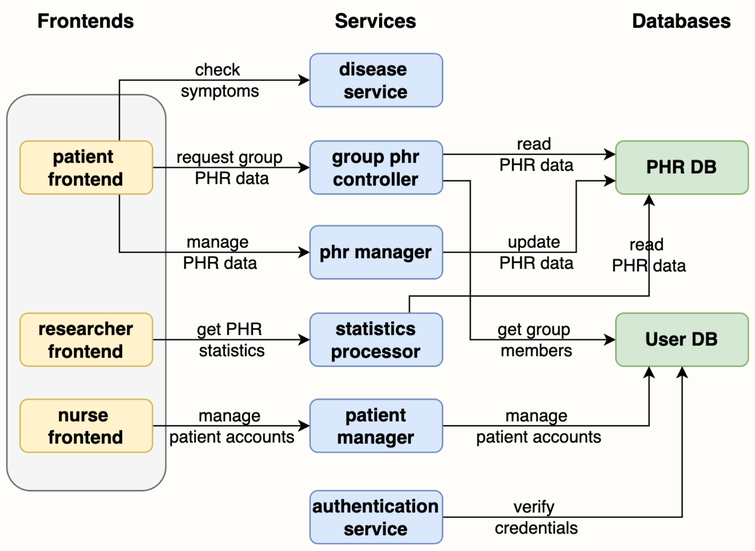


Figure 1: The overall architecture of the PCE, including the combined frontends (yellow), the backend microservices (blue), and the databases (green). Connections to the authentication service are made from most components, but are left out for better readability.

We follow two general goals in our implementation: First, we aim at **covering as many types of privacy threats as possible** defined by LINDDUN GO and second, we aim at including a **diverse set of technologies**, e.g. different programming languages, to prevent bias on any specific technology.

Table 1: An overview of the microservices in the application with short descriptions and their languages.

Service	Description	Language
frontend	The UI consisting of the three subcomponents patient-, researcher-, and nurse-frontend.	TypeScript
auth	Authentication backend which issues authentication tokens for the different roles (e.g. nurse, patient)	Go
disease service	Can be queried with symptoms to retrieve a list of possible diseases.	JavaScript
phr-manager	Allows patients to upload their Patient Health Records (PHR) to track their disease including medication and symptoms.	Python
group phr controller	Allows patients to query PHR of their group members to compare their course of disease, as well as medications and symptoms.	Python
nurse-api	Allows the registration of new patients and their assignment to a group by nurses.	Java
statistics	Allows researchers to retrieve statistics about PHR. It implements k-anonymity to protect patients' privacy.	Python
User DB	Holds patient names and the patients' group assignments	PostgreSQL
PHR DB	Holds Patient Health Records.	MongoDB

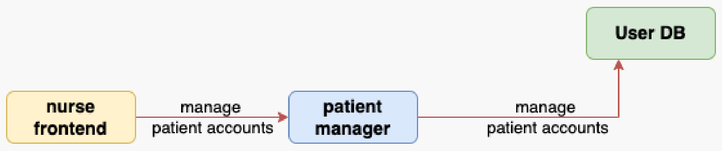
$\Longrightarrow$ Patients' medical data is at risk due to the service provider as well as other patients

Implemented Weaknesses

To enable the detection of privacy threats in our test bed, we implement privacy weaknesses in popular programming languages (like Python, Java and Go), which can be meaningfully represented in source code (e.g side-channel threats cannot be directly reflected in code). In total, 27 of 35 threats of the LINDDUN GO categorization [4] are implemented and also named according to the corresponding categories. In the following, three implemented example weaknesses are explained in more detail. The complete list can be found on the open-source project site on GitHub [5].

Threat 1: Identifying credentials (ID1)

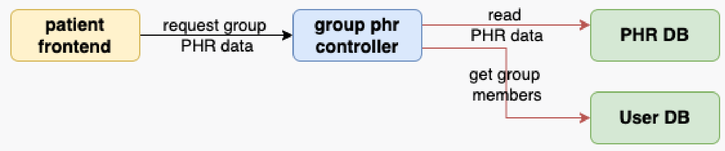
Patients are registered with identifiers (first name and last name).



Entry Point
frontend/src/NewUser.tsx: The new user data (incl. identifiers) are introduced by the nurse.
Exit Point
nurse-api/src/.../UserController.java: The user is created by the patient manager service.

Threat 2: Linkability of retrieved data (L7)

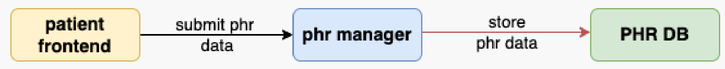
Retrieved personal data are linkable.



Entry Point
frontend/src/GetGroupPhrForm.tsx: A user requests PHR about fellow group members.
Exit Point
group-phr-controller/app.py: the group-phr-controller accesses the User DB and the PHR DB and links the pseudonymous PHR data to the users' identifiers

Threat 3: Detectable at storage (D4)

An API allows the detection of entities stored in a DB.



Entry Point
frontend/src/PhrForm.tsx: The user submits PHR, and can specify a custom user ID and group ID.
Exit Point
phr-manager/app.py: The PHR manager returns an error if the user is not member of the specified group, leaking information about which user is (not) member of a group.

Automatic analysis tools should be able to detect such threats, for example detect HTTP API responses and the protocols that are used to transmit data.

Conclusions

Summary: We provide a test bed as a standard for comparison of analysis tools, and a resource for data privacy education. We also hope to start a discussion about the possibility to detect privacy threats automatically, e.g. regarding code, policies, and side-channels.

Future Work: Implement further weaknesses, add synthetic data generation to facilitate real-time testing, and develop static application security testing tools.

Acknowledgement: This work was funded by the European Union Horizon 2020 project MEDINA, Grant No. 952633.

References

[1] Open Web Application Security Project (OWASP). Damn Vulnerable Web Applications directory. owasp.org/www-project-vulnerable-web-applications-directory/

[2] Paul E Black and Paul E Black. 2018. Juliet 1.3 test suite: Changes from 1.2. US Department of Commerce, National Institute of Standards and Technology.

[3] Patient Community system - Example Privacy analysis. Kim Wuyts. https://www.linddun.org/downloads, Patient communities example.

[4] LINDDUN GO: A lightweight approach to privacy threat modeling. Kim Wuyts, Dimitriy Van Landuyt, Laurens Sions, Joosen Wouter. In *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*.

[5] **Immanuel Kunz, Angelika Schneider, Christian Banse, Konrad Weiss, Andreas Binder. Patient Community Example implementation.** github.com/clouditor/patient-community-example