Comenius University in Bratislava

Faculty of Mathematics, Physics and Informatics

Design and implementation of an RFID access control system

Bachelor's Thesis

2016

Kamila Součková

Comenius University in Bratislava

Faculty of Mathematics, Physics and Informatics

Design and implementation of an RFID access control system

Bachelor's Thesis

Študijný program:	Informatics
Študijný odbor:	2508 Informatics
Školiace pracovisko:	Department of Computer Science
Školiteľ:	RNDr. Richard Ostertág, PhD.

Bratislava, 2016

Kamila Součková

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	
Študijný program:	informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor:	9.2.1. informatika
Typ záverečnej práce:	bakalárska
Jazyk záverečnej práce:	slovenský

Názov:

Cieľ:

Literatúra:

Kľúčové slová:

Vedúci:	
Katedra:	FMFI.KI - Katedra informatiky
Vedúci katedry:	doc. RNDr. Daniel Olejár, PhD.

Dátum zadania:

Dátum schválenia: doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

....
študent vedúci práce

Acknowledgements: TODO Jerry, Ostertág, Vinko, ŠVT/Adam, F za re-views?

Abstract

Abstract in the English language (translation of the abstract in the Slovak language).

Keywords:

Abstrakt

Slovenský abstrakt v rozsahu 100-500 slov, jeden odstavec. Abstrakt stručne sumarizuje výsledky práce. Mal by byť pochopiteľný pre bežného informatika. Nemal by teda využívať skratky, termíny alebo označenie zavedené v práci, okrem tých, ktoré sú všeobecne známe.

Kľúčové slová: jedno, druhé, tretie (prípadne štvrté, piate)

Table of Contents

Introduction

Project Deadlock is a system that controls access to a number of points of access (eg. doors, appliances) using RFID cards.

Unlike existing commercial solutions, Deadlock is fully open-source and open-hardware, and designed to be flexible, maintainable, and cost-effective. We provide tools and expose all interfaces and components, making Deadlock easy to integrate with existing systems and customize to the needs of the user.

Deadlock is designed for security and reliability, assuming untrusted and unreliable network.

Deadlock is a joint project by the faculty's Student Development Team[1] – implemented by students and managed by professors at the faculty.

This thesis first introduces the requirements/specification (chapter 1) and the high-level design choices we made to fulfill it (chapter 2). We then focus on the server/controller communication protocol (chapter 3) and the server design and implementation (chapter 4). Then we look at the future plans (chapter 5).

TODO mention which is my work and which is overview; talk to someone about Adam's thesis.

[1]http://svt.fmph.uniba.sk

Chapter 1

Specification

Project Deadlock aims to create a complete system to allow ISO/IEC 14443a-compatible cards (commonly known as *RFID cards*), such as International Student/Teacher Identification Cards, to be used to unlock doors and access other electronic appliances (hereafter *points of access*).

For this system to be useful at our university, Deadlock must meet the requirements outlined below.

1.1 Reliability

Points of access should be accessible even when things go wrong, specifically partial power or network outages must not make controllers stop allowing access nor lose access logs. Server failure must also cause no problems.

Furthermore, allowing for a simple implementation of server failover would be a good idea.

1.2 Security

As Deadlock may be used to protect valuable resources, such as computer rooms or labs, it must allow access if and only if it should.[1] Logs or card IDs may be private, so

[1]See 1.8.1 for the discussion of power outages.

they must not leak. Deadlock will be employed in publicly accessible places, meaning we cannot assume a private communication channel. Therefore all communication in both directions must be secret and authenticated.

1.3 Extensibility

In order to be prepared for the future, and also to make incremental development possible, all software and all hardware must be modular, with well defined interfaces, and extensible.

Functions not implemented in the first iteration, but expected to be added in the future, are

- arbitrary communication with the card,
- controlling arbitrary appliances, not just door locks,
- WiFi module (for cases when power is available but Ethernet is not).

1.4 Ease of development

In the future Deadlock will likely be developed and maintaned by students, not fulltime developers. Therefore the codebase must be simple, easy to understand and change, the tools and libraries must be easy to use, and the overhead of introducing a new developer to the project must be minimal.

1.5 Ease of use

Setting up access rules should be simple and convenient. Synchronization with the university's electronic information system is required, so that card info and groups like "CS teachers" or "PhD students" can be imported automatically.

It should bother a human if and only if human intervention is required – simple tasks and predictable issues should be handled automatically.

1.6 Ease of deployment and maintenance

Replacing any failed components should be quick and should not require substantial training.

Deployment should be simple and with minimal overhead. On the hardware side, it should be possible to leverage existing infrastructure in order to not need extra cables for communication or power. On the software side, importing data from existing sources (such as our university's Academic Information System) should be possible.

The system should check its state and automatically fix whatever can be fixed automatically, e.g. reboot a device if it gets locked up.

1.7 Availability

Hardware parts should be cheap to manufacture and components for them should either be available in the future or painlessly replaceable by their newer alternatives.

In order to make Deadlock as available as possible, we will release both the hardware schematics and the software to the public under the MIT license.

1.8 Further considerations

1.8.1 Power outage behavior

In case of a power outage, some doors should stay locked (to avoid the risk of breaching security), and some doors should open (e.g. emergency exits). While both can be supported, our use case requires only the "default close" behavior and therefore the current controller model is hard-wired for this case.

1.8.2 Emergency open

The system must implement a "force open" command which will unlock the door. This is useful in emergencies (as long as power is available).

Chapter 2

Design overview

TODO perhaps more details.

TODO maybe change structure: specification => fun design choices => important decisions => overview. Maybe not and use this structure only for "my" stuff (chapters 4 and 3).

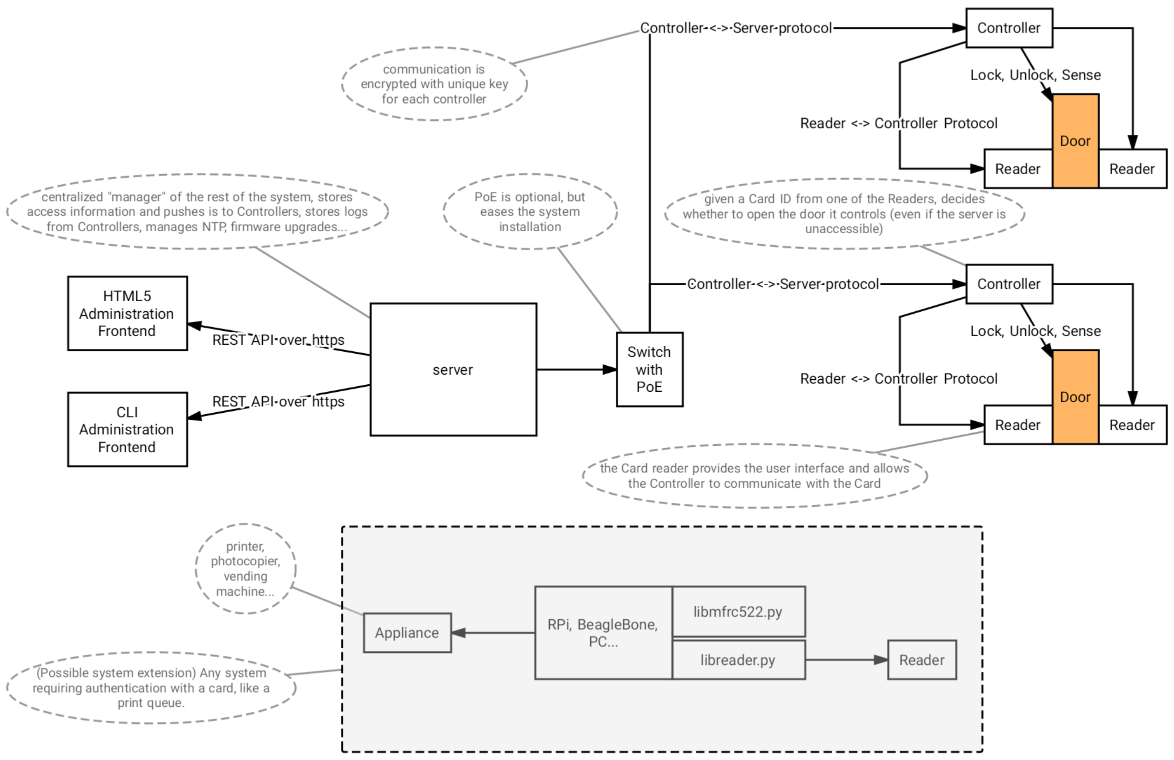
The system consists of a server and a number of controllers. Each controller serves a single point of access, holding a copy of the access rules and evaluating them locally. The server provides controllers with rules updates and collects access logs. We provide a management+monitoring UI.

2.1 Main components

2.1.1 Server

The server holds the authoritative version of the access rules, collects logs and provides software updates and time synchronization for the other devices. It monitors system state (and reports it to the management UI).

It is stateless – requests are served based on just the rules and logs in the database. This simplifies the code and makes replication and failover trivial.



Controller-<->-Server-protocol
Controller
Lock, Unlock, Sense
Reader <-> Controller Protocol
Door
Reader
Reader
communication is encrypted with unique key for each controller
centralized "manager" of the rest of the system, stores access information and pushes is to Controllers, stores logs from Controllers, manages NTP, firmware upgrades...
PoE is optional, but eases the system installation
given a Card ID from one of the Readers, decides whether to open the door it controls (even if the server is unaccessible)
HTML5 Administration Frontend
CLI Administration Frontend
REST API over https
REST API over https
server
Switch with PoE
Controller-<->-Server-protocol
Controller
Lock, Unlock, Sense
Reader <-> Controller Protocol
Door
Reader
Reader
the Card reader provides the user interface and allows the Controller to communicate with the Card
printer, photocopier, vending machine...
(Possible system extension) Any system requiring authentication with a card, like a print queue.
Appliance
RPi, BeagleBone, PC...
libmfrc522.py
libreader.py
Reader

Figure 2.1: Deadlock components. Note: this picture is horrible, **TODO** .

It is hardware-agnostic – it runs on anything with networking and a Python environment.

2.1.2 Controller

The controller controls its associated access point (e.g. unlocks its door). It takes actions (opening the door, logging) based on events observed (a card being presented, the door opens). It periodically pings the server, checking for updates.

The controller is "almost stateless" – logs are sent to the server, and rules and firmware updates can be retrieved from the server. Therefore a device can be swapped simply by writing the correct device ID and encryption key to either the device or the database.

2.1.3 Reader

Several card readers may be attached to the controller. We provide a library to interface with our readers, so they can be used independently of our controller.

2.2 Access rules

The decision whether to grant access is a fuction of user identity, access point, date, time, and day of week. Rules are of the form

$$(\textit{identity, access point, time specification}) \rightarrow \textit{allow} \,|\, \textit{deny}.$$

Default is “deny”; if multiple rules match, a “deny” rule overrides any “allow” rules.

As a simplification, identities, access points and time specifications can be grouped (even recursively).

2.3 Technical Challenges

2.3.1 Reliability

Controllers must work during network failures (without losing access logs). Solved by storing and evaluating the rules on the controller and making the protocol stateless and idempotent, allowing the controller to retry operations until they succeed and making server failover trivial.

2.3.2 Security

The system must securely operate over untrusted networks, resisting passive and active attacks. Therefore communication is encrypted and authenticated using a device-specific key via the NaCl library [2]. Nonces and idempotence prevent replay attacks.

2.3.3 Easy deployment and maintenance

The system must not require separate communication infrastructure nor dedicated power supplies. We use ethernet and support the Power over Ethernet standard.

Adding and replacing devices must not require substantial training. Solved by making device configuration minimal and making swapping devices with pre-configured ones trivial.

Deadlock must be usable decades from now, therefore it must depend only on components, libraries and tools which are likely to stay. This requirement needed to be taken into account when designing the hardware and software.

Chapter 3

Server $\leftrightarrow$ controller communication protocol

TODO why => how

References: [1–3]

Chapter 4

Server: Design and Implementation

4.1 Design

TODO

References: [6]

4.2 Implementation

TODO

References: [4, 5]

Chapter 5

Future plans

TODO plans, improvements, stuff

Conclusion

TODO Deadlock is awesome.

References

TODO make pandoc stop indenting these, argh

[1]Bernstein, D.J., Lange, T. and Schwabe, P. 2013. NaCl reference.

[2]Bernstein, D.J., Lange, T. and Schwabe, P. 2012. The security impact of a new cryptographic library. *Progress in cryptology–LATINCRYPT 2012*. Springer. 159–176.

[3]Postel, J. 1980. RFC 768: User datagram protocol. *http://www.ietf.org/rfc/rfc768.txt*. (1980).

[4]Python Software Foundation 2015. Python 3 documentation.

[5]Stufft, D. and Contributors, I. 2013. PyNaCl reference.

[6]Ullman, J.D. 1984. *Principles of database systems*. Galgotia publications.