Dizajn a implementácia RFID prístupového systému

Bakalárska práca
2016

Kamila Součková

viedol RNDr. Richard Ostertág, PhD.

DEADLOCK

vyvíjaný ŠVT*

moja práca:

- server
- komunikačný protokol
- časť architektúry

* http://svt.fmph.uniba.sk

POŽIADAVKY

- **dôveryhodnosť**
  - **spoľahlivosť**: zvládať chyby (HW aj SW)
  - **bezpečnosť**: zabrániť nepovolenému prístupu a úniku informácií
- **praktickosť**
  - **rozšíriteľnosť**: zmeny musia byť lacné
  - **jednoduchý na vývoj**: lacné udržiavanie; noví členovia
  - **jednoduchý na použitie**: prístupové práva; automatika
  - **jednoduchá inštalácia a udržiavanie**
  - **prístupnosť**: lacný; otvorený
- **iné**

} mimo rozsah

ŠKÁLOVANIE

- identity: **~ 10^5 - 10^6**
- prístupové body: **~ 10^3**
- pravidlá: ...čo sú pravidlá?
  - pridané manuálne: **~ 10^2 - 10^3**
- pre daný prístupový bod: výrazne menej
  - < 10^2 pravidiel
  - 1 prístupový bod :-)
- zmeny by sa mali rozšíriť do niekoľkých minút

VS. EXISTUJÚCE RIEŠENIA

	bežné	Deadlock
centrálny server	✓	✓
nezávislosť od výrobcu	✗	✓
rozšíriteľný	✗	✓
funguje pri nedostupnom serveri	✗	✓
navrhnuté s dôrazom na bezpečnosť	✗	✓
cena na 1 prístupový bod	~$300	$40–$60*
hotový	✓	✗ (zatiaľ)

* najmä vďaka použitu lacnej pracovnej sily (študentov)

PREHĽAD ČASTÍ SYSTÉMU

SERVER

- ukladá prístupové pravidlá
- zbiera záznamy o prístupe
- poskytuje aktualizácie SW a presný čas
- monitoruje stav systému

CONTROLLER (+ ČÍTAČKY KARIET)

- ovláda príslušný prístupový bod
  - napr. odomyká dvere
- koná na základe pozorovaných udalostí
  - napr. otvorí dvere, keď je priložená karta
- lokálna databáza prístupových pravidiel
  - nemusí sa pýtať servera
- hlási stav, pýta si aktualizácie pravidiel a FW
- posiela na server záznamy o prístupe
- “plug and play” (takmer žiadna konfigurácia)

ADMINISTRATÍVNE ROZHRANIA

- komunikujú so serverom cez HTTP API

- CLI pre skripty a ľudí, ktorí majú radi CLI
- nástroje na import dát (napr. z AISu)
- webové rozhranie:
  - editor prístupových práv
  - administrácia prístupových bodov
  - monitorovanie systému (v reálnom čase)

otázka č. 1:

PRÍSTUPOVÉ PRAVIDLÁ

VŠEOBECNOSŤ VS. POHODLNOSŤ

pozorovanie:

Za všeobecnosť sa platí zložitosťou.

- väčšina pravidiel danej inštalácie bude rovnakého tvaru ⇒ všeobecnosť vedie k zbytočnému, otravnému opakovaniu
- ALE: výnimky sa vždy nájdu ⇒ nemôžeme obetovať všeobecnosť

VŠEOBECNOSŤ VS. POHODLNOSŤ

pozorovanie:

Za všeobecnosť sa platí zložitosťou.

⇒ 2-úrovňový prístup:

- nízkoúrovňové/interné: jednoduché, expresívne
- vysokoúrovňové: špecifické pre účel
  - žiadne predpoklady o iných (nesúvisiacich) pravidlách v systéme, nesmú ich pokaziť
- každé (interné) pravidlo označené **ruleset**-om
  - vynucuje “nechytať nesúvisiace pravidlá”

TVAR INTERNÝCH PRAVIDIEL

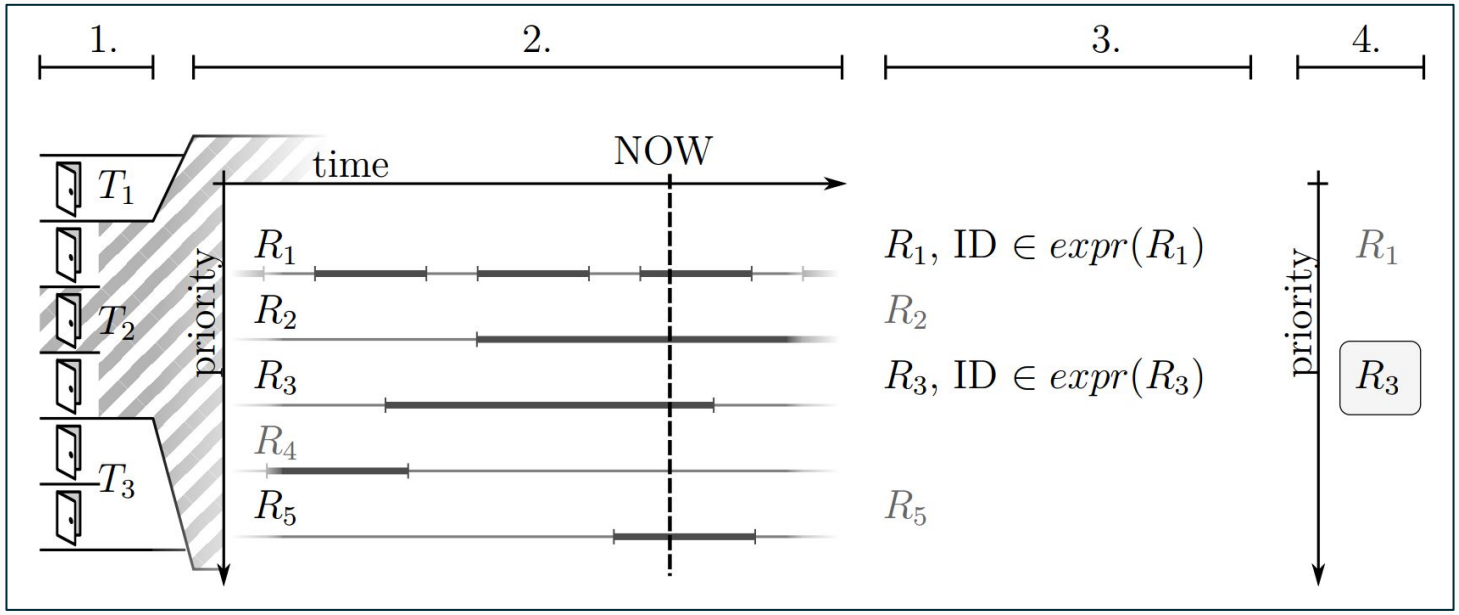
- prvý nápad: Boolovský výraz nad identitami, prístupovými bodmi, špecifikáciami času
  - logika na vyhodnocovanie a všetky dáta musia byť na controlleri

---

- každý prístupový bod je práve jedného ***typu***
- pre každý typ: pravidlá priradzujúce čas a ***výraz nad identitami*** k *“povol”* / *“zakáž”*
- zoradené podľa ***priority*** (rôzna)

VYHODNOCOVANIE PRAVIDIEL

1. vyber pravidlá pre typ daného prístupového bodu
2. z nich vyber pravidlá sediace na daný čas
3. z nich vyber pravidlá o danej identite
4. z nich vyber pravidlo s najvyššou prioritou (jedno)



VÝRAZY NAD IDENTITAMI

- zovšeobecnenie skupín
- (obmedzený) Boolovský výraz (len) nad identitami
- AND, OR, NOT (alebo podobné): NOT potrebuje uložiť komplement, alebo hack
- ⇒ INCLUDE, EXCLUDE (množinové zjednotenie a rozdiel); $\varnothing \setminus \{x\} = \varnothing$
- môže byť vyhodnotené nezávisle
  - jednoduchšie (pre ľudí aj počítače)
  - veci sa dajú predpočítať (“*id* patrí do výrazu *e*”)

Otázka č. 2:

KOMUNIKAČNÝ PROTOKOL

AKO A PREČO

- odolnosť voči chybám ⇒ viac serverov, málo stavu
- bezstavový + idempotentný
- controller: požiadavka → server: odpoveď
- požiadavky sú vždy nezávislé ⇒ netreba mať 1 server alebo synchronizáciu
- retransmisia: cyklický výber servera
  - ⇒ vždy rovnaké riešenie chýb pri komunikácii

AKO A PREČO

- odolnosť voči chybám ⇒ viac serverov, málo stavu
- bezstavový + idempotentný
- controller: požiadavka → server: odpoveď
- požiadavky sú vždy nezávislé ⇒ netreba mať 1 server alebo synchronizáciu
- retransmisia: cyklický výber servera
  - ⇒ vždy rovnaké riešenie chýb pri komunikácii
- aktualizácia za behu:
  - chyba ⇒ ďalší server – aj pri parsovaní správy
  - nasadiť server so starou aj novou verziou… to je všetko!

ČO

- **`PING`**: hlásenie stavu, “niečo nové?”, čas
- **`ALOG`**: prenos záznamov o prístupe
  - odpoveď OK ⇒ dáta sú zapísané na disku
- **`XFER`**: prenos časti súboru
  - explicitný začiatok a dĺžka (⇒ bezstavové)
  - explicitná verzia
    - garancia: rovnaká verzia ⇒ rovnaký obsah
- **`CRITICAL`**: kritický problém, treba niečo robiť
- **`ASK`**: Mám teraz pre túto identitu povoliť?
- **`ECHOTEST`**: testovanie a vstavané monitorovanie

kódovanie: CBOR (http://cbor.io)

BEZPEČNOSŤ

- bezpečnosť je ťažšia, ako si myslíš, aj keď vieš, že bezpečnosť je ťažšia, ako si myslíš
  - prvá verzia zraniteľná: replay attack
- ⇒ NaCl*: `secret_box(nonce, key, payload)`
  - jediná požiadavka: nonce sa použije najviac raz
    - tu: náhodná (24 bajtov ⇒ pravdepodobnosť kolízie >50% po $>10^{28}$)
- vyžaduje: bezpečný server, controller; verejný kanál
- zaručuje:
  - tajnosť, integritu, odolnosť voči analýze časovania (NaCl)
  - odolnosť voči replay útokom (idempotencia + nonce)

* https://nacl.cr.yp.to

Otázka č. 3:

SERVER: dizajn a implementácia

ORGANIZÁCIA DÁT

- chceme jednoduchosť a triviálnu replikáciu ⇒ závisieť len na DB
  - ostatné (pamäť, FS) sú len cache
  - DB má mechanizmy na replikáciu
- DB je viac ako len úložisko:
  - trigger-y ⇒ predpočítavanie “patrí do výrazu”
    - v budúcnosti bude inkrementálne
  - trigger-y + LISTEN/NOTIFY ⇒ hlási serveru, keď sa stane niečo zaujímavé

HLAVNÉ KOMPONENTY

- **`deadserver`**: komunikuje s controllermi
- **`deadapi`**: HTTP API pre zvyšok sveta
  - udalosti: NOTIFY z DB sa propaguje klientom ako prúd udalostí (EventSource/server-sent events)
- **`deadaux`**: pomocné práce
  - vytvára lokálne databázy pre controllery
  - jednoduchý “strážny pes” pre `deadserver`
- **rozhranie pre “spoločné súbory”**
  - abstrahuje “súbor”: meno, verzia, bajty
  - jediné spoločné rozhranie (okrem DB)

ZHRNUTIE

- **dôveryhodnosť**
  - **spoľahlivosť**: zvládať chyby (HW aj SW)
  - **bezpečnosť**: zabrániť nepovolenému prístupu a úniku informácií
- **praktickosť**
  - **rozšíriteľnosť**: zmeny musia byť lacné
  - **jednoduchý na vývoj**: lacné udržiavanie; noví členovia
  - **jednoduchý na použitie**: prístupové práva; automatika

ZHRNUTIE

- **dôveryhodnosť**
  - **spoľahlivosť**: lokálna DB, prechod na záložný server ✓
  - **bezpečnosť**: komunikácia poriadne autentifikovaná a šifrovaná ✓
- **praktickosť**
  - **rozšíriteľnosť**: dobrý návrh, modularita ✓
  - **jednoduchý na vývoj**: dobrý návrh, modularita ✓
  - **jednoduchý na použitie**: všeobecné + špecifické prístupové práva; monitorovanie + automatizácia ✓

ĎAKUJEM ZA POZORNOSŤ!

- **dôveryhodnosť**
  - **spoľahlivosť**: lokálna DB, prechod na záložný server ✓
  - **bezpečnosť**: komunikácia poriadne autentifikovaná a šifrovaná ✓
- **praktickosť**
  - **rozšíriteľnosť**: dobrý návrh, modularita ✓
  - **jednoduchý na vývoj**: dobrý návrh, modularita ✓
  - **jednoduchý na použitie**: všeobecné + špecifické prístupové práva; monitorovanie + automatizácia ✓

APPENDIX

OBSAH

POSUDOK ŠKOLITEĽA

- špecifikácia času:
  - čas (od do)
  - dátum (od do)
  - dni v týždni
  - NULL matchuje čokoľvek
  - prienik a pod. na úrovni pravidiel
- nastavovanie času vs. latencia: nepodstatné (do rádovo 100ms)
- poslanie príkazu zo servera: preferovaná možnosť je pridať príkaz k odpovedi na PING

POSUDOK ŠKOLITEĽA

- XFER: v bajtoch; mimo rozsah ⇒ chunk dĺžky 0 a EOF; chunk je vždy prítomný
- CRITICAL:
  - ako záznam do logu nestačí, pretože môže byť treba napr. upozorniť vrátnika
  - nedostupný server ⇒ opakovanie (rovnako ako vždy)
- ECHOTEST: request body je to, čo vždy, t.j. vnútro obálky; aj rozparsuje, takže overí, či funguje kódovanie

POSUDOK ŠKOLITEĽA

- sekvenčné nonces: môže byť, ale ťažšie zaručiť použitie len raz (čo ak premažem to, kde si píšem, čo bolo naposledy?)
- vysokoúrovňové pravidlá nesmú meniť niečo, čo nepoznajú, ale mali by to zobraziť užívateľovi (ako nízkoúrovňové pravidlá)
- priorita: v práci píšem, že je to niečo, čo určuje usporiadanie a je jednoznačné, takže prirodzene celé čísla (a rôzne)

POSUDOK ŠKOLITEĽA

- idempotencia vs. korekcia času: teoreticky sa môže stať, že 2 prístupy budú zapísané len raz, ale nie je to pravdepodobné
- HTTP API zatiaľ nie je finálne; CLI je samodokumentujúca (`--help`)
- ERDs: takto je to čo najznormalizovanejšie (napr. spojiť by vyrobilo časté NULL); ruleset je stĺpec v tabuľke rule; stĺpec “reader” môžem pridať, ak to bude potrebné

POSUDOK ŠKOLITEĽA

- algoritmus na výpočet `in_expr`: zdola nahor rekurzívne bublem zmeny
- FNV-1a: netestovali sme, testovali iní*; ak bude treba, je možné zmeniť, keďže o hashi nie sú žiadne predpoklady
- Deadlock nie je celkom hotový – ale práca, ktorá bola urobená, je netriviálna a to, čo chýba, je vďaka dobrému dizajnu jednoduché doplniť

* http://programmers.stackexchange.com/a/145633

POSUDOK OPONENTA

- správy v protokole
  - typ integer a jeho kódovanie: CBOR podporuje integery ľubovoľnej veľkosti, pričom spôsob kódovania je určený veľkosťou kódovaného čísla (a volí sa automaticky) – ako popísané v štandarde, §2.1
  - čas: štandardný timestamp, teda sekundy od 1.1. 1970 (veľkosť opäť aká treba, pričom implementácie interne používajú 64-bitový čas)
  - tag-y: “New entries in the range 256 to 18446744073709551615 are assigned by First Come First Served” – nie je problém

POSUDOK OPONENTA

- pravidlá
  - omylom je priložený starší ER diagram – ruleset je stĺpec v tabuľke rule (typu integer) a v priloženom zdrojovom kóde je
  - čas: nešpecifikované (NULL) matchuje na čokoľvek; musí sedieť všetko (AND)
  - link na dokumentáciu (GitHub wiki) je na stránke projektu, na ktorú je link v práci; novšia, podrobnejšia dokumentácia sa pripravuje na https://deadlock.readthedocs.io (a bude na ňu link zo stránky projektu, keď bude pripravená)

POSUDOK OPONENTA

- pretečenie času: tento problém neexistuje (ľubovoľne dlhé čísla v protokole; 64-bitový čas na serveri aj v controlleroch)
- ako zistiť, ktorá verzia je najnovšia: controller nijako; server vie (pri vygenerovaní novšej updatne “latest” symlink), v odpovedi na PING uvádza vždy najnovšiu
  - existuje možnosť pridať metadáta na začiatok blob-u
  - môže nastať “preskakovanie” pri rozsynchronizovaní serverov, ale to sa dá opraviť (“lenivým” controllerom)

POSUDOK OPONENTA

- príklady vysokoúrovňových pravidiel:

  univerzita: (napríklad – ale zanalyzujeme bežné prípady)

  "dvere *X* sú *učebňa* pre *všetkých* študentov *informatiky*"

  ⇒ typ "ucebna_inf_vsetci",
  expr. INCLUDE (*studenti_inf_vsetci*, *pedagogovia_vsetci*),
  čas *pracovné dni 8:00–19:00*

  hotel:

  "na izbe č. *X* bývajú ľudia s kartičkami (*Y1*, *Y2*, *Y3*)"

  ⇒ typ "izba_*X*", expr. INCLUDE (*Y1*, *Y2*, *Y3*), čas vždy
  typ "izba_*X*", expr. INCLUDE (upratovačky), čas ráno

KEY DESIGN PRINCIPLES

- **modularity**
  - independent modules with well-defined, simple, minimal interfaces
- **Principle of Least Astonishment**
  - “People are part of the system. The design should match the user’s experience, expectations, and mental models.” * (also for developers)
- **state is bad**
  - state ⇒ complexity and difficulties with failure recovery

* Saltzer, J.H. and Kaashoek, M.F. 2009. Principles of computer system design: an introduction.

RULESET

- every rule tagged with exactly one **ruleset**
- ruleset is atomic for write operations
- why:
  - helps ensure consistency
  - high-level rules applications touch only their rulesets
    - may be enforced in DB via row-level security

PACKET FORMAT

“record”: key-value map with a few pre-defined keys

⇒ CBOR encoding (as [(tag, value)] ⇒ key-value map):

- self-describing (meaning + type encoded)
- expressive: data types, nesting
- forward-compatible: new field ⇏ parse error
- embedded-friendly: small code size, fast
- standard, with existing libraries