Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

CAPTCHA

2019

Bc. Kristián Vraštiak

Vedoucí práce: doc. RNDr. Miroslav Kolařík, Ph.D.

Studijní obor: Aplikovaná informatika, prezenční forma

Bibliografické údaje

Autor:	Bc. Kristián Vraštiak
Název práce:	CAPTCHA
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2019
Studijní obor:	Aplikovaná informatika, prezenční forma
Vedoucí práce:	doc. RNDr. Miroslav Kolařík, Ph.D.
Počet stran:	78
Přílohy:	1 CD/DVD
Jazyk práce:	slovenský

Bibliograhic info

Author:	Bc. Kristián Vraštiak
Title:	CAPTCHA
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2019
Study field:	Computer Science, full-time form
Supervisor:	doc. RNDr. Miroslav Kolařík, Ph.D.
Page count:	78
Supplements:	1 CD/DVD
Thesis language:	Slovak

Anotácia

Práca prehľadovo popisuje, čo je CAPTCHA a k čomu slúži. Predstavuje známe prístupy rozlišovania legitímnych používateľov od robotov, silné a slabé stránky prístupov. Nahliada do niektorých súčasných implementácií. Výstupom implementačnej časti práce sú štyri vlastné CAPTCHA testy. Súčasťou je taktiež rozbor ich robustnosti voči prelomeniu útočníkmi.

Synopsis

This work briefly describes what CAPTCHA is and what it is for. It presents known approaches to distinguishing legitimate users from robots, strengths and weaknesses of those approaches. The work looks into some of the current implementations. The output of the implementation part of the thesis are four own CAPTCHA tests. It also includes an analysis of their robustness against attackers.

Kľúčové slová: CAPTCHA; hádanka; obrana voči robotom; spam; javascript

Keywords: CAPTCHA; puzzle; defense against robots; spam; javascript

Ďakujem doc. RNDr. Miroslavovi Kolaříkovi, Ph.D. za odborné vedenie tejto diplomovej práce.

Čestne vyhlasujem, že som celú prácu vrátane príloh vypracoval/a samostatne a za použitia iba zdrojov spomínaných v texte práce a uvedených v zozname literatúry.

dátum odovzdania práce podpis autora

Obsah

Zoznam obrázkov

Zoznam zdrojových kódov

1 Úvod

Internet sa od svojho masívneho rozšírenia v 90. rokoch veľmi rýchlo stal najrozsiahlejším zdrojom informácií. S obrovským počtom webových služieb a voľnosťou prispievať na rôzne fóra, komentovať články, čítať viac-menej verejne dostupné osobné informácie zo sociálnych sietí, sa neprekvapivo objavilo množstvo spôsobov, ako spomenuté činnosti zneužiť. Pre široký rozsah a rýchly postup sa problémom stalo najmä automatizované zneužívanie služieb. Za zneužitie, s ktorým sa stretol už pravdepodobne každý, sa dá považovať napríklad inzercia pochybného produktu nejakými spamovacími robotmi (spambotmi) na fóre alebo v inej forme internetovej diskusii a samozrejme, mailom.

Technológia CAPTCHA je spôsob, ako rozoznať automatizovaného útočníka od legitímneho používateľa. Používa sa v prípadoch, keď chceme zabrániť prístupu útočníka – robota ku konkrétnym internetovým zdrojom. V dnešnej dobe existuje množstvo druhov CAPTCHA fungujúcich na rôznych princípoch, každý z nich má svoje výhody a nevýhody.

Cieľom tejto diplomovej práce je prehľadovo popísať technológiu CAPTCHA, jej históriu, súčasný stav. Popísať silné a slabé stránky niektorých existujúcich CAPTCHA. Ďalej taktiež vymyslieť niekoľko nových CAPTCHA a popísať ich slabé a silné stránky.

1.1 CAPTCHA - história

Návrh Turingovho testu spočíva v tom, že počítač tento test úspešne prejde, ak pod určitými podmienkami dokáže reagovať podobne ako človek. Pôvodný návrh počíta s troma terminálmi obsluhovanými účastníkmi, vzájomne od seba fyzicky oddelenými. Jeden účastník je počítač, ostatní sú ľudia. Test prebieha nasledovným spôsobom: jeden z ľudských účastníkov podáva otázky. Ostatní dvaja účastníci (teda počítač a človek) odpovedajú. Otázky su kladené v špecifickom kontexte a formáte. Po čase alebo po vyčerpaní otázok má spytujúci sa za úlohu rozhodnúť, ktorý z odpovedajúcich je človek. V tomto smere sa CAPTCHA od originálneho Turingovho testu líši tým, že rozlíšiť stroj od človeka má za úlohu počítač.

Prvá zmienka automatizovaného Turingovho testu pochádza z nepublikovaného rukopisu izraelského vedca Moniho Naora. Rukopis obsahoval kľúčové myšlienky k tejto problematike, no žiaden návrh na samotný automatizovaný test, či jeho formálnu definíciu. Prvá podoba technológie CAPTCHA, ako ju poznáme dnes, boja vyvinutá v roku 1997. Išlo o prepis textu z obrázku, ktorý je nejakým spôsobom zdeformovaný, niekedy navyše zakrytý inými písmenami alebo číslicami. Akronym CAPTCHA znamená „Completely Automated Public Turing test to tell Computers and Humans Apart“, je to teda akýsi obrátený Turingov test rozlíšenia medzi človekom a strojom. CAPTCHA sa od originálneho Turingovho testu líši aj tým, že testované subjekty zvyčajne nerozprávajú, ale využívajú svoje zmysly – najčastejšie zrak a sluch. Pôvodný Turingov test bol

jedine konverzačný. Prvý krát bola CAPTCHA verejne použitá vo vyhľadávači AltaVista na ochranu proti automaticky registrujúcim sa robotom.

Patenty z rokov 1997 a 1998 priamo nedefinujú výraz CAPTCHA. Detailne však popisujú myšlienky a princípy využívané v technológiách CAPTCHA dodnes. A to využívanie zmyslových a kognitívnych schopností na riešenie jednoduchých problémov, osvedčene mimoriadne zložitých pre počítačový program. Medzi takéto schopnosti patrí spracovanie zmyslových informácií, ako identifikovanie objektov alebo textu v graficky rušivom prostredí.

V praxi je CAPTCHA systém generujúci a vyhodnocujúci testy schopné rozlíšiť človeka od stroja, ktoré väčšina ľudí úspešne splní a roboti v týchto testoch zlyhajú. Potreba rozoznávať robotov od skutočných ľudí je kvôli narastajúcemu počtu robotov zahlcujúcich webové služby komerčnými propagáciami. Majitelia webových služieb teda siahajú po technológiách schopných prevažnú väčšinu týchto nežiadaných činností zastaviť. Technológiu CAPTCHA je možné využiť ako:

- Zabránenie automatizovanej distribúcii spamu: uistí, že email alebo príspevok odoslaný webovej službe je zaslaný človekom, nie robotom
- Zabránenie automatizovaným registráciám: blokuje robotov pokúšajúcich sa registrovať množstvo nových účtov pre webové služby poskytujúce napríklad email, fóra, sociálne siete, a pod.
- Zabránenie automatizovaným distribúciám spamu na sociálnych sieťach
- Zabránenie slovníkovým útokom proti prihlasovacím formulárom: a iným útokom hrubou silou
- Zabránenie manipulácii hlasovaní a súťaží: zaistí, že v online hlasovaniach hlasujú iba skutoční ľudia
- Zabránenie zberu prostriedkov: znemožňuje dolovaniu virtuálnych prostriedkov, zahŕňajúc i dolovanie predmetov v online hrách

Existuje množstvo implementácií CAPTCHA. Niektoré rozlišujú používateľov od robotov klasicky pomocou zdeformovaného textu, často je taktiež položená otázka a z niekoľkých obrázkov má následne používateľ za úlohu vybrať ten správny. Rôznym druhom implementácií sa budem podrobnejšie venovať v kapitole 2. Napriek veľkému množstvu už existujúcich riešení má aj dnes zmysel sa touto problematikou zaoberať. Inteligencia a schopnosti robotov sa totiž časom zdokonaľujú, reverzné vyhľadávanie obrázkov už dnes ponúka nejeden vyhľadávač, musia sa teda zdokonaľovať aj mechanizmy na obranu proti robotom. S dobou sa taktiež mení správanie používateľov na internete: používatelia sú zvyknutí na rýchle načítanie stránok, stávajú sa netrpezlivejšími, k webovým službám sa pripájajú z rôznych zariadení. Napriek rôznorodosti implementácií, každá CAPTCHA by mala spĺňať niekoľko kritérií, aby bola reálne použiteľná a kvalitná.

Používateľ by mal stráviť minimum času riešením CAPTCHA. Vypĺňanie zložitého CAPTCHA testu môže používateľov frustrovať, o to viac ak ho nakoniec zloží neúspešne a musí ho odznovu zopakovať. Nie zriedkavo je taktiež nesprávne vyplnenie CAPTCHA testu penalizované spôsobom nútiacim používateľa zložiť test úspešne viackrát. Je pravdepodobné, že CAPTCHA nesplňujúca toto kritérium odradí príliš veľké množstvo legitímnych používateľov, ktorí dajú prednosť inej službe.

Vyplývajúc z účelu a motivácie technológie CAPTCHA, je efektívnosť v tomto prípade udávaná ako pomer úspešných riešení legitímnych používateľov a nesprávnych riešení poskytnutých robotmi. Žiadaná je aspoň 90% úspešnosť používateľov a neúspešnosť robotov by mala byť čo najbližšia nule. Keďže robotmi ľahko prelomiteľná CAPTCHA je neefektívna, s efektívnosťou súvisí aj neprelomiteľnosť, ktorej sa budem venovať v sekcii 1.2.

Jednoduchosť: testom by mal prejsť každý gramotný človek bez nejakých špecifických odborných poznatkov. Teda napríklad žiadne vedomostné testy, riešenie sústav rovníc, a pod.

Prístupnosť: webové služby sú využívané aj handicapovanými ľuďmi. Využívajú zariadenia pre slabozrakých, Braillove tlačiarne a písacie stroje, rôzne čítačky. Pre pohybovo postihnutých ľudí existujú vstupné zariadenia reagujúce na pohyby. Je preto žiadané, aby CAPTCHA nediskriminovala handicapovaných ľudí, čo sa spomína aj vo výnose č.55/2014 o štandardoch pre informačné systémy verejnej správy, kde v bode 1.1.5 Prílohy č.1 stojí „Ak je obrázok použitý na odlíšenie či webovú stránku ovláda človek, alebo počítač, používateľovi sú k dispozícii i také metódy, ktoré umožňujú toto odlíšenie uskutočniť napriek neschopnosti používateľa získať požadovanú grafickú informáciu z obrázka, napríklad zvukový výstup, doplňovanie bežného textu a podobne. Na odlíšenie človeka a počítača sa používa plne automatizovaný verejný Turingov test na rozlíšenie počítačov od ľudí, ktorým je metóda CAPTCHA a podobne.“[1] V Českej republike podľa vyhlášky č. 64/2008 o prístupnosti platí „Je-li obrázek použit kvůli odlišení, zda se stránkou pracuje skutečný člověk nebo počítač (tzv.CAPTCHA), jsou uživateli k dispozici i doplňkové metody, které umožňují toto odlišení provést.“[2] To samozrejme predstavuje ďalší netriviálny problém: ku klasickému testu pridať ďalší test, ktorý bude môcť byť vyriešený iným zmyslom než predošlý test, no zároveň by tento test mal byť pre robota približne rovnako náročný na prelomenie. V opačnom prípade ponúknutie pre robota ťažkého testu spolu s pre robota jednoduchým testom môže viesť k zneužitiu testu určeného pre handicapovaných.

[1] http://jaspi.justice.gov.sk/jaspidd/vzory/014055Pr1.pdf

[2] http://www.mvcr.cz/clanek/metodicky-pokyn-k-vyhlasce-c-64-2008-sb-o-forme-uverejnovani-informaci-souvisejicich-s-vykonem-verejne-spravy-prostrednictvim-webovych-stranek-pro-osoby-se-zdravotnim-postizenim-vyhlaska-o-pristupnosti.aspx

1.2 Robustnosť

Útoky alebo pokusy o prelomenie CAPTCHA rozpoznávania je možné rozdeliť do nasledujúcich kategórií. CAPTCHA je robustná, ak pre útočníka nie je výhodné pokúšať sa o prelomenie žiadnym z prvých troch z týchto spôsobov:

- Hádanie
- Používanie známych odpovedí
- Strojové učenie
- Ľudské zdroje

Najjednoduchším spôsobom pokusu o prelomenie CAPTCHA je hádanie. Útočník sa jednoducho podľa implementácie požadovaného CAPTCHA náhodne pokúsi uhádnuť správnu odpoveď. Úspešnosť tejto techniky závisí od typu testu. Najľahšie prelomiteľné testy touto technikou sú také, kde má používateľ za úlohu vybrať správnu z niekoľkých možných ponúknutých možností. Naopak, šanca prelomenia testov typu, kde možnosti ponúknuté nie sú, napr. prepisu zdeformovaného textu, či vypočítania matematického výrazu, je metódou hádania značne nízka.

Útok na princípe používania známych odpovedí využíva obmedzené množstvo dát na strane poskytovateľa CAPTCHA testu, resp. jeho neschopnosť vygenerovať unikátny test pre každú overovaciu požiadavku. Opakujúce sa testy môže totiž útočník využiť a to ukladaním si správnych odpovedí a pri opätovnom dotaze útočník odpovie už správnou odpoveďou z jeho databázy.

Strojové učenie je najpokročilejším a najefektívnejším spôsobom útoku, najmä pre obrázkové typy testov. Táto kategória predstavuje napríklad metódy rozpoznávania a analýzy obrazu účinné pri obrázkových CAPTCHA testoch.

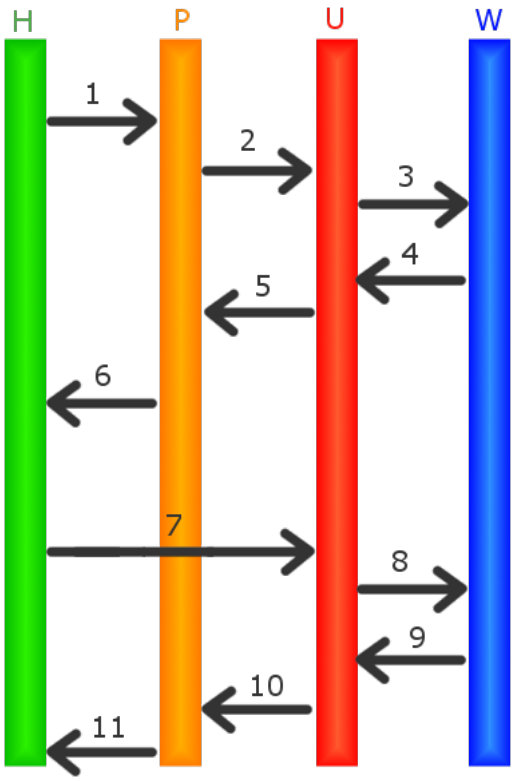
CAPTCHA testy sú založené na otvorených problémoch umelej inteligencie, napríklad dekódovanie a následne rozpoznanie skreslených obrázkov a textu. Preto CAPTCHA testy taktiež vytvárajú výzvy pre výskumníkov v tejto oblasti. Tieto testy sú výhodné pre oba smery: buď je CAPTCHA prelomená a istý problém umelej inteligencie je vyriešený alebo CAPTCHA prelomená nie je a existuje spôsob, ako rozlíšiť človeka od stroja.

Aj tá najdokonalejšia CAPTCHA sa dá jednoducho prelomiť ak sú na jej prelomenie, v tomto prípade je vhodnejšie povedať vyriešenie, použití ľudia. To nie je nutné brať ako nedostatok alebo slabinu, keďže prirodzene CAPTCHA má rozlíšiť človeka od stroja. Problém je však masíve zneužívanie ľudských zdrojov, predovšetkým z krajín tretieho sveta, kde za vyriešenie tisíc testov dostane človek niečo málo nad 1 euro. Jednou z najznámejších služieb riešenia testov CAPTCHA pomocou lacných ľudských zdrojov je spoločnosť „Death by Captcha“. Podľa oficiálnej stránky tejto spoločnosti sa priemerné vyriešenie jedného testu pohybuje od 7-12 sekúnd s úspešnosťou 93% (aktualizované každú minútu). CAPTCHA test je zaslaný pomocou im ponúkaného API náhodnému pracovníkovi, ktorý test

následne vyrieši. Pracovníci sa nachádzajú v juhovýchodnej ázii, Indii a v Pakistane[3]. Služba je podľa portálu ponúkaná 24 hodín denne, dokáže vraj vyriešiť ruské alebo všeobecne cudzojazyčné testy. Riešitelia CAPTCHA testov pre útočníkov nemusia byť len zamestnanci špecializovanej firmy. V prípade útoku presmerovaním to môžu byť aj bežní používatelia, ktorí o tom zväčša nevedia. Každý test rozlíšenia človeka od stroja útočník presmeruje na skutočného človeka, ktorý využíva inú webovú službu sľubujúcu za vyriešenie testu výsledok. Útočník riešenie testu ponúkne webovej službe, ktorá test vygenerovala. Tento útok je nazývaný aj „pornografický útok“, keďže riešitelia sú často návštevníci stránok takéhoto typu [1]. Obrázok 1 demonštruje myšlienku takéhoto útoku.

1. Používateľ H chce využiť služby stránky P, ktorá spolupracuje s útočníkom U.
2. Služba P informuje útočníka.
3. Útočník U sa snaží o prístup k webovej službe W.
4. Tá na jeho požiadavku odpovedá HTTP odpoveďou obsahujúcou CAPTCHA.
5. Útočník prepošle CAPTCHA test médiu P.
6. P oznámi používateľovi, že mu sprístupní svoj obsah po tom, čo vyrieši CAPTCHA, zašle mu ju.
7. Používateľ H test vyrieši a odošle útočníkovi. (môže prebiehať aj cez P)
8. Útočník sa ňou pokúsi autorizovať
9. W vyhodnotí riešenie a odošle odpoveď U.
10. V prípade, že bolo riešenie správne, U o tom informuje službu P. V opačnom prípade sa prejde ku kroku 2.
11. P poskytne používateľovi H sľúbený obsah.

[3] http://www.deathbycaptcha.com/user/compensation



Obr. 1: Útok presmerovaním.

1.3 Druhy CAPTCHA

Podľa typu úlohy poskytnutej na vyriešenie bude v tejto práci rozlišených niekoľko druhov CAPTCHA testov. Táto kapitola popíše niekoľko druhov najpoužívanejších CAPTCHA. CAPTCHA testy sú pre verejné použitie poskytované aplikačným rozhraním, kde autorská aplikácia reaguje na požiadavku o vygenerovanie novej CAPTCHA samotnou úlohou a riešenia niekedy v podobe hashu. Ak by bolo riešenie poskytované v nekódovanej forme, bolo by jednoduché pre útočníkov zbierať správne odpovede prostými požiadavkami. Hashovanie odpovedí nemá význam vždy. Množina správnych odpovedí pre CAPTCHA test často nie je dostatočne obsiahla, aby sa nedalo hrubou silou (hashovaním všetkých možností) zistiť, o ktorú odpoveď ide. Z tohto dôvodu nie sú v práci poskytované odpovede hashované.

1.3.1 Rozpoznávanie textu

Prvý, ale stále hojne používaný druh, je založený na prepise textu. Takéto testy spočívajú na princípe vygenerovania obrázku s textom, ktorý používateľ musí správne prepísať do textového poľa a potvrdiť. Aby takáto CAPTCHA nebola jednoducho prelomená OCR technikami, text je značne zdeformovaný. Takéto

deformácie sú napríklad znaky otočené o nejaký uhol, prekrývajúce sa písmena, náhodné čiary alebo krivky cez text. Vďaka zdokonaľujúcim sa OCR technikami rastie tiež aj miera deformácie textu často hraničiaca s nečitateľnosťou človekom. Podrobnejší popis v sekcii 2.2.2.

1.3.2 Rozpoznávanie obrázku

Obrázková CAPTCHA predstavuje alternatívu ku klasickej textovej. Najčastejšia implementácia obrázkového druhu CAPTCHA spočíva v položenej otázke a ponúknutých niekoľko možností vo forme obrázkov. Pre používateľa je táto forma CAPTCHA testu rýchlejšia, pohodlnejšia a jednoduchšia, otázka je, či je aj efektívnejšia. Nevýhoda tejto formy je nutnosť poznať popis obrázkov, ktoré sa volia pri vytváraní CAPTCHA testu. Popis sa dá získať buď priamym priradením človekom alebo zväčša menej presne – strojovým priradením. Pre implementácie tejto formy CAPTCHA sú typické tieto charakteristiky robustnosti:.

- Hádanie: používateľ si často volí z 9 až 25 obrázkov, čo činí túto formu CAPTCHA testu pomerne jednoducho napadnuteľnú prostým zvolením náhodného obrázka. Možné riešenie by mohla byť potreba vyriešiť viacero takýchto testov.

- Používanie známych odpovedí: obmedzená množina dát je v tomto prípade skôr množina obrázkov, ku ktorým poznáme presný popis. Priradzovanie popisu ľuďmi je pomalšie, čo má za následok obmedzenejšiu bázu dát, no zároveň taktiež strojové priradzovanie generuje väčšie množstvo chybných alebo nepresných popisov.

- Strojové učenie: techniky rozpoznávania obrazu dnes už rozpoznať zviera alebo tvár dokážu. Ďalšou možnosťou, ako dostať popis k obrázkovej možnosti je reverzné vyhľadávanie obrázkov.

1.3.3 Logické

Úloha je predaná používateľovi v textovej forme a odpoveď sa odosiela buď taktiež v textovej forme, alebo pomocou možností. Logické typy spočívajú v zodpovedaní jednoduchej textovej otázky. Otázky môžu byť vedomostného (napríklad „Čo loví mačka?“) alebo matematického charakteru („koľko je $100+100$?“). Otázok vedomostného charakteru, dosť jednoduchých na to, aby ich správne vedelo zodpovedať minimálne 90% používateľov, nie je dosť veľké množstvo. Útočník si teda všetky možné otázky dokáže zozbierať rýchlo spolu so správnymi odpoveďami.

Ďalšou nevýhodou je nutná znalosť jazyka, v akom je otázka položená. Naopak, jednoduché je poskytnutie alternatívy pre handicapovaných ľudí, keďže položenú otázku stačí prečítať programom, ktorý to dokáže.

1.3.4 Zvukové

Generátor CAPTCHA testu ponúkne zvukový záznam hovorených slov. Podobne, ako pri textových formách, aj tu sa informácia najprv zdeformuje. Aj v tomto prípade šumom, no niekedy aj inými pridanými zvukmi, ktoré by nemali brániť v interpretácii pôvodnej zvukovej informácie človekom. Používateľ si zvuk prehrá a zadá do textového poľa slová, ktoré počul. Aj tu je nevýhodou nutná znalosť jazyka, v akom je zvukový záznam vygenerovaný, ak je obsahom zvuku hovorená reč. Ďalšou nevýhodou môže byť neschopnosť stroja prehrať zvuk (napr. desktopový PC bez reproduktorov) alebo nevôľa človeka púšťať zvuk nahlas, najmä ak sa nachádza na verejnosti.

1.3.5 Testy nevyžadujúce interakciu

Táto forma nepožaduje po používateľovi žiadnu priamu interakciu s rozpoznávacím systémom – namiesto toho rozpoznávanie prebieha na základne používateľovho správania sa. Automatizovaný útok na webové služby môže prebiehať vykonaním skriptu, ktorý je založený na automatickom prechádzaní stránok a interpretácii HTTP odpovedí. Vyplní formuláre, následne ich odošle a celý postup zopakuje. Spôsobov, ako rozpoznať robota plniaceho skript je niekoľko:

- Požadovanie sedenia: keďže odoslanie vyplneného formuláru je len HTTP požiadavka typu POST (alebo GET), útočník, ak pozná formát požiadavky, ju môže zaslať bez predošlého načítania HTML dokumentu. Server teda musí detegovať, že HTML obsah nebol zobrazený, čo odlišuje stroj od človeka, keďže legitímny používateľ odošle HTTP požiadavku pomocou formulára zobrazeného v HTML dokumente. Server vygeneruje session kľúč, ktorý odošle spolu a pripojí sa k formuláru HTML dokumentu. HTTP požiadavka, ktorá sa odošle, už session kľúč mať bude. Požiadavky z formuláru odoslané bez session kľúča alebo s nezhodujúcim sa kľúčom nebudú prijaté.

- Honeypot: tento spôsob spočíva v nasadení formulára, ktorý je v skutočnosti atrapa. Ak je útočníkov skript jednoduchý a CSS neinterpretuje, tak tento formulár vyplní a odošle, čím sa od používateľa odlíši. Ten totiž formulár nevidí: formulár je skrytý pomocou „display: none“ CSS atribútu alebo JavaScriptom zo stránky odstránený, resp. iným spôsobom skrytý. Uvedený spôsob nie je dostatočne robustný. Nasadiť na web robota ušitý na mieru nie je problém. Pre útočníkov dnes rovnako není problém takéto prvky odignorovať interpretovaním kaskádových štýlov alebo iným spôsobom.

- Prihlásenie účtom sociálnej siete: dnes čoraz častejší spôsob overovania. Nie úplne bez interakcie – avšak interakcia sa pôvodne vyžaduje za iným účelom, napríklad pridávanie komentárov. Používateľ si overenie ani nemusí uvedomovať. Toto ale nie je úplna náhrada za CAPTCHA test[2]. Nehodí

sa totiž všade: formuláre, kde má zmysel požadovať prihlásenie s účtom sociálnej siete, je naozaj málo.

1.3.6 Filter

Pri použití tohto spôsobu server obsahuje filter zachytávajúci správanie, ktoré je podľa vopred definovaných pravidiel označené za podozrivé. Existuje niekoľko prístupov:

Zakázané slová: prístup využíva fakt, že niektoré slová majú omnoho väčšiu šancu objaviť sa v príspevkoch robotov, než legitímnych používateľov. Statické uloženie týchto slov a následné filtrovanie príspevkov obsahujúce dané slová nie je robustné podľa druhého kritéria – používanie známych odpovedí. Robot sa jednoducho týmto kľúčovým slovám bude vyhýbať. Bayesovo filtrovanie je na rozdiel od statického spôsobu schopné prispôsobiť sa meniacemu sa správaniu robotov. Je založené na myšlienke, že ak podreťazec nájdeme v spame (automatizovane odoslanom robotom), ale nie v príspevkoch od legitímnych používateľov, tak algoritmus usúdi, že príspevok je pravdepodobne spam. Nevýhodou je, že filter tieto poznatky pred nasadením nemá a trvá mu, kým sa ich naučí. Model sa trénuje explicitným označovaním nežiadaných príspevkov. Pre každé slovo si filter prispôsobí údaj o pravdepodobnosti výskytu slova v nežiadaných príspevkoch. Slová používané v legitímnych príspevkoch vo filtri naopak vzbudzovať podozrenie nebudú. Po naučení pravdepodobností slov (jedná sa vlastne o funkcie vierohodnosti) sú použité na rozhodnutie, či príspevok bude považovaný za nežiadaný, alebo či je v poriadku. Na zvýšenie presnosti môže byť vhodné kombinovať Bayesovo filtrovanie s predefinovanými pravidlami.

Black list: čierna listina rozsahu IP adries, z ktorých majú roboti tendenciu pripájať sa. Nevýhodami tohto prístupu je dočasnosť riešenia, ako aj možné blokovanie legitímnych používateľov.

1.3.7 Autentifikácia

Problém overenia, či ide o človeka alebo stroj, by sa dal preadresovať na autentifikačné služby. Prispievať obsah sa povolí iba prihláseným používateľom, ktorí sa registrujú pomocou nejakej autentifikačnej služby. Autentifikačná služba pri registrácii overí, či ide o človeka a konkrétne, o ktorého človeka ide. To služba zaistí napríklad zaslaním kódu pomocou SMS správy, ktorý má používateľ prepísať. Ak by sa neidentifikovala konkrétna osoba, nijak by sa registrácia od klasického CAPTCHA testu nelíšila.

1.3.8 Biometrika

S vývojom technológií, ako aj s pokrokom automatizovaných útokov môžme v budúcnosti očakávať sofistikovanejšie metódy overovania pravosti používateľov. Tak, ako sme nedávno zaznamenali príchod pokročilejšieho spôsobu overenia

od Googlu jednoduchým kliknutím vedľa textu „Nie som robot“, môžme tiež očakávat, že sa v blízkej budúcnosti objaví využívanie biometriky v každodennom živote. Dnešné telefóny už bežne disponujú senzorom na detekciu odtlačkov alebo zabudovaným softwarom na rozpoznávanie tvárí. Síce nie úplne ako CAPTCHA test, ale už v dnešnej dobe sa používa biometrika pri overovaní identity používateľa, napríklad pri nákupoch. Je možné, že v blízkej budúcnosti to už nebude slúžiť ako náhrada za heslo, ale aj ako plnohodnotný CAPTCHA test[2].

2 Rozbor vybraných súčasných implementácií

2.1 TextCaptcha

Implementácia TextCaptcha[4] je CAPTCHA ponúkajúca logické úlohy v textovej forme. API umožňuje odpovedať na požiadavky generovania testu vo forme XML (kód 1) a JSON (kód 2).

```
<captcha>
 <question>If tomorrow is Saturday, what day is today?</question>
 <answer>f6f7fec07f372b7bd5eb196bbca0f3f4</answer>
 <answer>dfc47c8ef18b4689b982979d05cf4cc6</answer>
</captcha>
```

Zdrojový kód 1: Textcaptcha výzva XML formáte

```
{
"q": "If tomorrow is Saturday, what day is today?",
"a":["f6f7fec07f372b7bd5eb196bbca0f3f4",
  "dfc47c8ef18b4689b982979d05cf4cc6"]
}
```

Zdrojový kód 2: Textcaptcha výzva v JSON formáte

V odpovedi je otázka v atribúte „q“ pre JSON a v elemente question pre XML formát. Pole všetkých možných správnych odpovedí možno nájsť v atribúte „a“, elemente answer. Takúto odpoveď možno dostať HTTP požiadavkou pre adresu v tvare http://api.textcaptcha.com/<yourID>.<format>, kde <yourID> je identifikátor služby využívajúcu toto API a <format> je XML alebo JSON formát. Teda pre http://api.textcaptcha.com/56595.json dostávame odpoveď:

Teda úlohou je zodpovedať, ktoré z čísel 44, 8 a 4 je najmenšie. Podporovaný jazyk je (britská) angličtina, takže odpoveď by mohla byť Four alebo 4. Podľa API treba používateľovu odpoveď zbaviť prebytočných medzier na začiatku a konci reťazca, konvertovať všetky znaky na malé a následne vytvoriť

[4]dostupná na http://textcaptcha.com/

```
{
"q":"44, 8 or 4: which of these is the lowest?",
"a":[ "a87ff679a2f3e71d9181a67b7542122c",
  "8cbad96aced40b3838dd9f07f6ef5772"]
}
```

MD5 hash. A skutočne, prvý hash uvedený v poli odpovedí zodpovedá MD5 hashu reťazca „4“ a druhý „four“.

Možné typy logických úloh, ktoré používateľ má vyriešiť:

- aká je n-tá cifra v čísle m?
- z daných slov, koľko z nich je časť tela?
- z daných slov, koľko z nich je farba?
- dané číslo napísané slovami prepíšte číselne
- vyriešte $n@m$, kde n,m sú prirodzené čísla a @ je jedná z operácií plus alebo mínus daná slovne alebo znakmi
- dnes je (deň v týždni), aký deň v týždni bude zajtra? + variácie miesto dnes – včera, resp. zajtra
- z daných slov, ktoré z nich je deň víkendu?
- z daných čísel, vyberte najmenšie/najväčšie
- z daných slov, ktorá je v poradí n-tá farba?
- z daných slov, ktorá je v poradí n-tá časť tela?
- z daných slov, ktoré je v poradí n-té číslo?
- farba predmetu je? (farba predmetu je v texte zmienená)
- ako sa volá osoba? (meno osoby je v texte zmienené)

TextCaptcha nedisponuje príliš dobrej robustnosti. Otázky typu koľko častí tela alebo koľko farieb sa nachádza v nasledujúcich slovách nie sú ťažko uhádnuteľné, keďže vymenovaných slov pre pohodlie používateľa nezvykne byť veľký počet – 1 až 6. Z 50 vygenerovaných otázok bolo pre 7 možností odpoveď „1“, ktorá bola aj najčastejšou odpoveďou, čo predstavuje 14% šancu na úspech pri súčasnom odpovedaní „1“. Má zmysel teda po používateľoch požadovať viacero správnych odpovedí.

2.1.1 TextCaptcha solver

V práci bol navrhnutý a implementovaný skript, ktorý demonštruje, ako by mohlo vyzerať automatické riešenie TextCaptcha testu. Skript schopný riešenia týchto testov nazvem TextCaptcha solver. TextCaptcha solver slúži jedine na demonštračný účel; výstupom práce nie je skript automatizovane prelamujúci TextCaptcha. GUI softwaru je znázornené na obrázku 2.

Obr. 2: TextCaptcha solver.

Rozhranie má podobu jednoduchého HTML dokumentu s dvoma tlačidlami. Prvé tlačidlo „Get new and solve“ dostane z TextCaptcha API JSON odpoveď s otázkou a hashom správnej odpovede z adresy poskytovanej API http://api.textcaptcha.com/56595.json. Test vyrieši, otázku zobrazí pod titulok „TextCaptcha:“, odpoveď pod titulok „Answer:“, hash odpovede skontroluje s hashom správnej odpovede poskytnutým API a výsledok zhody či nezhody zobrazí pod titulok „Correct:“. Tlačidlo „Click to solve saved captchas“ vyrieši pevne zadané uložené pole 50tich takýchto CAPTCHA testov a vypíše ich prehľadne podobným spôsobom.

Všetka logika je implementovaná v jazyku JavaScript. Riešenie každého testu spočíva v dvoch fázach:

1. Rozpoznanie typu úlohy
2. Vyriešenie úlohy

2.1.2 Rozpoznanie typu úlohy

Na zistenie, o aký typ úlohy ide, používam zhodu s vhodne zloženými regulárnymi výrazmi a jednoduchý bodový systém. Každý typ úlohy dostane akési číselné vyhodnotenie šance, že ide o úlohu daného typu. V reťazci – otázke sa hľadajú kľúčové slová, ktoré by mohli naznačovať, že ide o danú úlohu. Napríklad vybrať zo slov, ktorá je v poradí n-tá farba je iná úloha, než spočítať, koľko farieb sa v otázke nachádza. Preto zhoda slova farba pridá body úlohám oboch spomínaných typov, no detekcia rádovej číslovky v otázke bodovo predbehne úlohu typu koľko farieb sa v otázke nachádza. Funkcie obsahujú aj alternatívne zadané úlohy, ktoré TextCaptcha API nevygenerovalo, no očakáva sa, že by mohlo: rádové číslovky sa medzi približne 100 vygenerovanými testami nachádzali jedine v tvare 1^{st}, 2^{nd}... , nikdy nie slovom, no bodovacie funkcie počítajú aj so slovným vyjadrením, podobne sa očakávajú alternatívy iných úloh. Nakoniec sa vyberie maximum z vypočítaných bodov a podľa toho sa rozhodne, ktorý problém sa bude riešiť. Ak sa zhoda nenájde, funkcia vráti nulové skóre, čo naznačuje, že nepôjde o úlohu daného typu. Predpokladajme, že v úlohe bude treba vybrať časť tela. Keďže časť tela sa v angličtine sa často vyjadruje dvoma spôsobmi(body part, part of the body), aj regulárny výraz počíta s oboma podobami. Toto overenie a následné zvýšenie bodov ale musí byť len za predpokladu, že bola nájdená aj rádová číslovka. Ak by predpoklad platiť nemusel, tak v prípade úlohy typu „koľko častí tela“ by sa tejto úlohe nesprávne zvýšilo skóre.

2.1.3 Vyriešenie úlohy

V tejto sekcii budú popísané princípy riešenia niektorých typov úloh. Majme napríklad najjednoduchšiu úlohu a tou je zistiť meno osoby, ktoré je v otázke uvedené. Príklad otázky takejto úlohy by mohol byť „The name of Lisa is?“. Pomocou regulárneho výrazu nájdeme všetky slová začínajúce sa na veľké písmeno a odpovieme, že meno je naše posledné nájdené slovo. To vráti správnu odpoveď pre vyššie uvedenú otázku, no aj pre jednoduchšie formy, ako napríklad „John's name is?“. Nakoniec odpoveď prevediem na reťazec malých písmen a zahashujeme.

Ďalšie znázornené riešenie bude riešenie typu úloh napr. „What is 8 add 10?“. Z textu vydolujeme zoznam čísel a operátorov, či už sú popísane slovne alebo inak. Teda napríklad z reťazca „one, 7 thousand and thirty two, 5“ vráti čísla 1, 7032 a 5. V poli s operátormi sa budú nachádzať všetky nájdené zhody regulárneho výrazu /minus|plus|add|+|-/[5]. TextCaptcha generuje len úlohy, kde je

[5]s príznakmi global a case insensitive

potreba sčítať alebo odčítať. Ak sa z nejakého dôvodu nenašiel ani jeden operátor, tipneme si výsledok 1. K prvému nájdenému číslu budeme pričítavať alebo odčítavať všetky ostatné. TextCaptcha generuje tento typ úlohy iba s dvoma prirodzenými číslami a jedným operandom, no implementácia ráta s viacerými číslami a operandmi.

Predpokladám, že čísel sa v reťazci bude nachádzať viac, preto ich bude potreba nejakým spôsobom oddeliť. Využijem to, že TextCaptcha oddeľuje čísla čiarkami a slovami „and“ a „or“. Taktiež ale treba počítať s tým, že čísla je možno oddeliť operátorom, napr. vo výraze „eight + 10“ sú čísla oddelené operátorom plus. V anglickej gramatike sa avšak správne pre čísla väčšie ako 100 píše v slovnom vyjadrení pred prvé slovo menšie ako sto spojka „and“. V tomto prípade ale spojka neoddeľuje čísla, preto je dôležité všetky spojky plniacu „gramatickú“ úlohu odstrániť a to správnym nahradením. Nahradenie funguje spôsbom: nájdi výskyt slova hundred, za ktorým môže nasledovať biely znaka následne podreťazec „and“ a celú zhodu nahraď reťazcom hundred. Podobne pre thousand a million. Potom sa už dané spojky môžu nahradiť čiarkami. Z reťazca „three thousand and twenty five and 45 or forty six“ sa teda stane „three thousand twenty five, 45, forty six“. Ostáva toto pole previesť na 3 čísla. To funguje nasledovným spôsobom: pre každý prvok poľa si zapamätám základ a sumu. Prechádza sa cez čísla. Ak je číslo menšie, než 100, do základu bude pričítané aktuálne číslo. V opačnom prípade pôjde o číslovku, ktorou bude možno treba násobiť. Pokiaľ je základ prázdny (teda, ak je 0), nebude sa číslom násobiť, ale do základu sa hodnota čísla pričíta. Ďalej, ak základ nie je prázdny, treba zistiť, či už pred aktuálnym číslom bolo väčšie číslo. Ak áno, znamená to, že násobiť sa bude základ, nie celá suma. Napríklad majme „two hundred thousand three hundred“, teda „2 100 1000 3 100“, aktuálne číslo 100. Pre prvú stovku platí, že $suma = 2 < 100$, teda bude sa násobiť celá suma. Suma bude po tejto iterácii 200. Pre druhú stovku ale platí $suma = 200000 > 100$, teda bude sa násobiť jedine základ, teda $3 \cdot 100$. Ak suma prevyšuje aktuálne číslo, po vynásobení aktuálneho čísla so základom sa výsledok pripočíta k sume. Ak aktuálne číslo prevyšuje sumu, znamená to, že predošlé hodnoty iba udávali násobok aktuálneho čísla, čiže 100, 1000 alebo 1000000 sa nimi vynásobí. Po tom, čo bol základ k sume pripočítaný, je treba základ vynulovať. V poslednom kroku cyklu ešte do sumy pripočítam zvyšný základ, pre prípad, kedy bolo posledné číslo menšie ako 100. Ešte raz si ilustrujme algoritmus na príklade „sixty nine thousand four hundred and seventy seven“. Po prevedení dostávame „60 9 1000 4 100 70 7“, s čím bude algoritmus pracovať nasledovne (i je iterácia, an je aktuálne číslo, $base$ je základ):

i	an	$base$	sum
0	60	60	0
1	9	69	0
2	1000	0	69000
3	4	4	69000
4	100	0	69400
5	70	70	69400
6	7	77	69477

2.1.4 Vyhodnotenie

TextCaptcha generuje pekne formátované otázky, s oddelenými možnosťami na výber a s malou variabilitou, na čo sa dá napísať na mieru skript, ktorý rieši takto zadané otázky. Z 50 otázok mal implementovaný riešiteľský algoritmus stopercentnú úspešnosť. Využíval už spomínané oddelené možnosti čiarkami a spojkami – text nebol nijak deformovaný. Premenlivosť otázok rovnakého typu napr.:

- What is one million and two as digits?
- Enter the number one thousand seven hundred and twenty eight in digits:
- What is forty thousand seven hundred and ninety four as a number?

nebola dostačujúca, na to aby sa skriptu nepodarilo identifikovať typ úlohy a následne ho vyriešiť.

Využívanie korektného a fixného formátovania v skripte riešiacom TextCaptcha testy má za následok veľký pokles v presnosti v prípade, že sa formát otázky zmení. Aj napriek tomu, že táto implementácia počíta aj s niektorými prípadmi, ktoré sa v TextCaptcha testoch nenachádzajú, ale v budúcnosti by mohli, zmena formátu otázok a ich deformácia by si vyžiadala zmenu rozpoznávacích a riešiacich algoritmov. Napríklad algoritmus detekcie mena funguje na princípe vybratia posledného slova, ktoré sa začína na veľké písmeno. V prípade, že by TextCaptcha generovala tieto typy úloh s každým slovom začínajúce sa na veľké písmeno, presnosť používateľov by pravdepodobne takmer vôbec neklesla, zatiaľ čo algoritmus použitý v skripte by odpovedal správne jedine v prípadoch, kde je meno posledné slovo — úspešnosť riešenia by drasticky klesla. To by sa riešilo buď fixnou databázou mien v skripte, alebo by algoritmus využíval existujúce databázy mien na internete.

Zoznam otázok vygenerovaných TextCaptcha API, ktoré boli vo finálnej verzii skriptu testované (niektoré otázky sa nemusia presne zhodovať s ich generovanou verziou):

- What is the third body part in yellow, thumb, toe, brick and wrist?
- What is the 1st digit in 7876391?

- Lion, tracksuit, rice, library, tooth and apple: how many body parts in the list?
- What is sixty nine thousand four hundred and seventy seven as a number?
- What is the 2^{nd} number in the list twelve, 9 and 21?
- six add 10 minus 15 is what?
- Yesterday was tuesday, what day is tomorrow?
- What day is tomorrow, if yesterday was Saturday?
- Saturday, Friday or Tuesday: which day is part of the weekend?
- What day is today, if tomorrow is Thursday?
- Three + eight equals ?:
- Enter the lowest number of 36, 14, 79, four, 54 or ninety nine:
- 26, sixty three, 68, 40, seventy seven or thirty two the largest is?
- Which digit is 7^{th} in the number 1077293
- Brown, hotel, green, elephant and blue: the 3rd colour is?
- Fourteen minus 3 equals ?
- What is one million and two as digits?
- The colour of a black nose is?
- Enter the number one thousand seven hundred and twenty eight in digits:
- What is forty thousand seven hundred and ninety four as a number?
- What is seventy eight thousand nine hundred and three as digits?
- If the duck is black, what colour is it?
- Of the numbers 53, 31 or twenty, which is the biggest?
- John's name is?
- The name of Lisa is?
- The number of body parts in the list head, chest, tooth, heart, elephant and T-shirt is?
- In the number 7235970, what is the 7^{th} digit?
- Eight add 10 is what?

- How many colours in the list foot, pink, nose and purple?
- The brown shirt is what colour?
- The list face, coffee and butter contains how many body parts?
- The list yellow, shorts, duck, ant, knee and wine contains how many colours?
- If a person is called Richard, what is their name?
- The list coat, chest, cow and white contains how many colours?
- Eight plus ten is what?
- One, thirty two, ten, twenty two and 33: the 4th number is?
- How many colours in the list trousers, fruit, hotel, green, apple and bank?
- What is two plus 7?
- How many colours in the list glove, church and black?
- What is eighty six thousand three hundred and ninety as digits?
- Of the numbers five, forty three, 52, fifty six or 51, which is the largest?
- What is thirty one thousand six hundred and thirty three as digits?
- 11, 25, 32, twenty four and nineteen: the 2nd number is?
- The colour of a green dog is?
- If the tracksuit is black, what colour is it?
- Rice, yellow, Red and black: How many colours in the list?
- The 4th number from 19, four, eighteen and twenty two is?
- The list coffee, horse and foot contains how many body parts?
- Enter the number forty nine thousand nine hundred and two in digits:
- 12, thirty seven, three and 25: the 2nd number is?
- Enter the number seventy thousand and eighty five in digits:

Množina farieb použitá v skripte obsahuje napríklad farby „purple“, „magenta“ alebo aj „beige“. Množina častí tiel použitá v skripte obsahuje napríklad „finger(s)“, „chest“ alebo aj „thigh(s)“.

2.2 Google reCaptcha

2.2.1 The NoCAPTCHA reCAPTCHA

V roku začal 2013 Google namiesto klasických obrázkových CAPTCHA nasadzovať testy sledujúce interakciu používateľa s CAPTCHA testom prostredníctvom internetového prehliadača. Dokáže v mnoho prípadoch rozoznať človeka od stroja sledovaním stôp nevyžadujúce priamu používateľovu interakciu. Nesplnenie (možné nesplnenie aj bez samotnej interakcie používateľa) tohto druhu znamenalo ponúknutie ďalšieho, resp. niekoľko ďalších CAPTCHA testov, tentokrát už obrázkového a náročnejšieho. Koncom roka 2014 už začal tento druh prevažovať nad ostatnými druhmi CAPTCHA testov používaných pri Google službách.

Tento druh testu sa používateľovi javil len ako odškrtávacie políčko [3] vedľa textu „Nie som robot“ (ang. verzia „I'm not a robot“). Používateľ už nemusí zápasiť s prepisovaním ťažko čitateľného textu, stačí na dané políčko kliknúť. To platí samozrejme len za predpokladu, že sa správanie testu nezdá nijako podozrivé. Používateľ si nemusí a väčšinou ani nie je vedomý, čo považuje tento test za podozrivé a čo, naopak, napovedá, že používateľ je legitímny. IP adresy, cookies uložené z ostatných webových domén a taktiež aj jeho pohyb myšou po ploche testu – to všetko slúži ako dôkaz legitímnosti. Google samozrejme tají presné sledované informácie. Ich zverejnenie by totiž mohlo pomôcť prelomeniu tohto testu a teda robotom dostať sa cez tento filter. Všetko, čo sa o tomto teste vie, bolo odpozorované alebo Googlom zverejnené (nie do podrobností). 60%-80% používateľom stačil 1 klik na overenie tohto druhu testu[6].

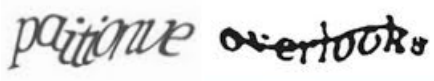
Pre používateľov používajúci mobilné zariadenia, Google ponúkne rovno zbierku obrázkov s úlohou rozoznať obrázky alebo naopak v rozdelenom obrázku označiť jeden predmet. Častou úlohou je napríklad zobrazenie obrázkov z premávky s úlohou označiť všetky vozidlá/autá/autobusy, atď.

2.2.2 Google reCaptcha – rozpoznávanie deformovaného textu

Ako bolo popísané v predošlej kapitole, tento druh CAPTCHA overuje legitímneho používateľa pomocou prepisu z obrázka obsahujúci zdeformovaný text. Súčasná implementácia Google reCaptcha ponúka tento druh testu až pri opakovaných dotazoch od jedného používateľa. Bežný používateľ by pri prvom overovaní mal klasicky vidieť prvý druh a to odkliknutie „Nie som robot“.

Neprelomiteľnosť rozpoznávania deformovaného textu z obrázka výrazne zvyšuje obtiažnosť samotného nachádzania znakov, než ich následné rozpoznávanie. Totiž segmentácia (lokalizácia jednotlivých znakov) je vo všeobecnosti ťažký a výpočtovo drahý problém. Preto kvalitná CAPTCHA tohto druhu by mala byť odolná voči segmentácii. Existujú rôzne mechanizmy odolné voči segmentácii, napríklad mechanizmy spájajúce susedné znaky, niekedy znaky až prekrývajúc. Spájajúce mechanizmy sa osvedčili a používajú sa dodnes.

[6] https://www.wired.com/2014/12/google-one-click-recaptcha/



Obr. 3: Zdeformovaný a pospájaný text reCaptcha

Okrem (proti) segmentačných algoritmov sa tento druh Google reCaptcha snaží bojovať proti prelomeniu ešte niekoľkými spôsobmi:

- líši sa hrúbka písma
- deformácia celého reťazca
- text je náhodný, nevytvárajúci slová
- použitých niekoľko rôznych typov písma

Existuje mnoho prístupov ako (automatizovane) riešiť/prelomiť túto variantu Google reCaptcha testu. V tejto práci ukážeme jeden zo spôsobov[4]. Možné prelomenie by sa mohlo skladať z 3 krokov:

- Predspracovanie obrázka
- Detekcia znakov pomocou vzorkovania
- Znaková segmentácia

2.2.3 Predspracovanie obrázka

Predspracovanie obrázka spočíva v prvotnom zväčšení rozlíšenia – zjemnení hrán jeho textu. Následne je obrázok konvertovaný na čierno-biely. Proces posterizácie na dve farby sa môže uskutočniť klasickým high-pass filtrom: pixely s farbou vyššou než stanovená hranica sú konvertované na čiernu, ostatné na bielu farbu. Posledným krokom v predspracovaní obrázka je zníženie hrúbky textu. To sa uskutoční z niekoľko dôvodov: normalizuje sa hrúbka písma, teda na hrúbke písma vo vstupnom obrázku nezáleží. Ďalším dôvodom je zrýchlenie výkonnosti v ďalších krokoch – pre spracovanie menšieho množstva pixelov.

Obr. 4: Obrázok po predspracovaní

2.2.4 Detekcia znakov pomocou vzorkovania

V tomto kroku sa deteguje, do ktorej zo štyroch kategórií znak spadá. Kategórie sú rozdelené podľa charakteristických tvarov znakov:

- Znaky s bodkami: i, j
- Okrúhle znaky: a, b, d, e, g, o, p, q
- Krížové znaky t, f
- Znaky tvaru S

2.2.5 Identifikácia znaku s bodkami

Na vzniknuté komponenty sa použije tzv. segmentáciu farebnou výplňou. Každý komponent sa momentálne môže skladať z jedného znaku, skupiny znakov alebo z časti znaku a všetky sú vďaka predspracovaniu čiernej farby a majú rovnakú hrúbku písma. Po segmentácii farebnou výplňou sa každý komponent zafarbí inou farbou

Obr. 5: Komponenty sú po segmentácii farebnou výplňou rozličnej farby

Znaky 'i' a 'j' sa skladajú z tela znaku a bodky nad ním. Detekcia bodky spočíva v spočítaní pixelov jedného komponentu – malý počet indikuje že ide o bodku. Pre detekciu tela je potreba najprv nájsť komponenty, ktoré sa nachádzajú pod bodkou. Znovu sa aplikuje algoritmus segmentácie farebnou výplňou, tentokrát taktiež prechádza všetkými smermi, no pri vypĺňaní ignoruje horizontálne susedné body – napodobňovanie tvarov písmen 'i' a 'j'. Cez novovzniknuté komponenty sa pretne pomyselná čiara tak, aby prechádzala cez bodku, pričom príliš vzdialené komponenty ignorujeme. Komponent najbližšie k bodke bude pravdepodobne telo znaku.

2.2.6 Identifikácia okrúhleho znaku

Na farbu pozadia obrázka sa aplikuje segmentácia farebnou výplňou. Po týchto krokoch sú komponenty pozadia oddelené rôznou farbou. Komponenty pozadia teraz vypĺňajú oblúky znakov, ale aj umelé oblúky vytvorené spájaním (segmentácie) znakov. Umelé oblúky sa dajú rozoznať od oblúkov znakov vďaka tomu, že počet vyplnených pixelov je malý, resp. je veľký, ale blízko sa nachádza oblúk znaku – to indikuje, že ide o umelo vytvorený oblúk vytvorený spájaním.

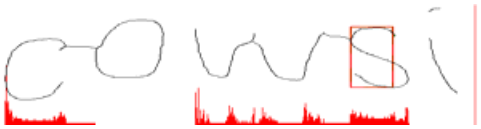
Obr. 6: Segmentácia farebnou výplňou aplikovaná na znakové a umelé oblúky

2.2.7 Identifikácia krížového znaku

Kreslením akéhosi imaginárneho obdĺžnika okolo krížovej časti znaku sa dosiahne, že ho znak pretne v každej strane práve raz. Táto skutočnosť sa dá využiť pri detekcii kríža v znaku: prechádza sa obrázkom s týmto imaginárnym obdĺžnikom a ak každá jeho hrana pretne práve jednu časť znaku s rovnakou farbou, tak aktuálna pozícia obdĺžnika indikuje, že sa tam môže nachádzať písmeno krížového tvaru. Po detekcii jednej možnej pozície sa všetky ostatné detekcie krížu v rámci už detegovanej pozície ignorujú. Takto sa pokračuje, kým nie je prejdený celý obrázok. Z konečných možných detegovaných znakov s krížom sa odstránia tie, ktorých detegovaný kríž sa nachádza v dolnej polovici komponentu – u ktorých teda asi nakoniec nepôjde o krížový znak. Taktiež tie, kde pixely znázorňujúce kríž nie sú súvislé a tie, kde sa kríž pretína s oblúkom iného znaku.

2.2.8 Identifikácia znaku tvaru S

Najprv je treba vytvoriť vertikálny histogram mapujúci počet pixelov farby inej, než je farba pozadia. Keďže znaky sa môžu pretínať, ignorujeme časti histogramu odpovedajúce pretínajúcim sa častiam. Ďalej sa v histograme nájde postupnosť za sebou idúcich hodnôt o 3 alebo viac pixeloch. Ak táto postupnosť je väčšia než nami stanovený prah dostačujúci na šírku písmena 's', oddelí sa tento znak od susedných, kde sa ako hranice použijú prvý a posledný stĺpec nájdenej postupnosti.



Obr. 7: Znázornenie počtu pixelov v stĺpcoch obrázka pomocou histogramu

2.2.9 Segmentácia

V tejto časti sa oddelia znaky detegované v predošlom kroku. Ako prvé sa vyplnia všetky detegované tvary znakov na bielo. To jednak skryje nájdený vzor a rozdelí spojené znaky na oddelené komponenty.

Ostávajúce viditeľné komponenty budeme rozlišovať ako vlastné a zdieľané. Vlastné komponenty patria jedine jednému znaku. Teda napríklad na Obr. 5 zelená a červená časť komponentu sú znaku vlastné, pretože patria len jednému

znaku. Zdieľané komponenty sú výsledkom spájania znakov, nepatria teda len jedinému znaku, na Obr. 8 je to napríklad je to oranžová časť, spája 'p' a 'i'.

Obr. 8: Zdieľaný komponent

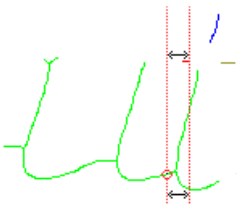
Medzi týmito dvoma druhmi sa rozlišuje nasledovným spôsobom: zdieľaný komponent spája viacero znakov a teda má oveľa väčší počet pixelov, než vlastné komponenty. Úloha segmentácie detegovaného znaku sa teda mení na zisťovanie, kde pretnúť zdieľaný komponent. Na hotový znak je potreba správne pretnutý zdieľaný komponent, detegovaný tvar znaku a jeho pričlenené vlastné komponenty.

2.2.10 Identifikácia pretínacích bodov

Poloha pretínaceho bodu zdieľaného komponentu záleží na znaku, s ktorým sa komponent spája. Zdieľaný komponent spájajúci znak s bodkou má pretínací bod (horizontálne) blízko, pretože takýto znak má malú šírku a ak by bol pretatý ďaleko od jeho vlastného komponentu, mohli by byť zničené znaky s ním spojené. Identifikácia pretínacieho bodu pre znaky krížového tvaru je podobná a pre rovnaký dôvod.

Pre zdieľaný komponent spájajúci znak okrúhleho tvaru, jeho pretínací bod je od neho ďalej. Okrúhly tvar sa totiž nachádza vo vnútri znaku, pretínanie blízko znaku by ho poškodilo. Podobne pre znaky tvaru 's'.

Na Obr. 9 je znázornené nájdenie pretínaceho bodu, kde zdieľaný komponent spája písmena 'u' a 't'. Znak 't' je znak krížového tvaru, preto jeho pretínací bod bude blízko neho. Červená kružnica znázorňuje pretínací bod a šípka znázorňuje jeho vzdialenosť od vlastného komponentu písmena 't'.



Obr. 9: Nájdenie pretínacieho bodu

2.2.11 Pretínanie

Body pretínania sa hľadajú v obrázku so stenčeným textom, no ozajstné pretínanie je výhodné uskutočňovať na pôvodnom obrázku. Jednou z výhod je znovupoužiteľnosť – už rozpoznané znaky sa môžu zhodovať s ďalšími v budúcnosti

ponúknutými. Ďalšia výhoda môže byť jednoduchšie a presnejšie vykonanie OCR po segmentácii.

Pri samotnom pretínaní sú body pretínania prenesené zo spracovaného do pôvodného obrázku. Súradnice pretínacích bodov budú v oboch verziách obrázku rovnaké, keďže rozmery obrázkov sú totožné. Do obrázku je následne vložený imaginárny obdĺžnik (na Obr. 10 je konkrétne rozmerov 6x15), s jeho stredom v bode pretínania. Vnútri obdĺžnika sa pretne znak najkratšou možnou cestou, ktorá ho dokáže pretnúť celý. Ak takáto cesta neexistuje, jednoducho pretneme znak vertikálne cez bod pretínania.

Obr. 10: Oddelenie zdieľaných komponentov

Obr. 11 znázorňuje výsledok pretínania zdieľaných komponentov v „hrubšej“ verzii obrázka, kde každý segmentovaný znak je zvýraznený rôznou farbou. Písmena už oddelené farbou je jednoduché izolovať napr. farebným filtrom. Takto izolované písmená je prosté prečítať/previesť na text pomocou OCR techník. Znaky získané z výsledkov OCR techník aplikovaných pre každé izolované písmeno sa zreťazia v správnom poradí, čím sa už získa správny výsledok.

Obr. 11: Výsledok segmentácie

Podľa autorov [4] dosahuje tento spôsob prelomenia úspešnosť 68% na testovacej vzorke 100 CAPTCHA testov. Všeobecný test na nezávislých 400 vzorkách dosiahol 62% úspešnosť. Úspešnosť je počítaná aj vrátane s implementáciou OCR technológie, ktorej úspešnosť sa pohybuje okolo 95%. Útok tvorci implementovali v jazyku Java na stroji s procesorom 2,4 GHz Intel Core 4 CPU a 4GB operačnou pamäťou. Priemerný čas potrebný na prelomenie jedného testu bol približne 7 sekúnd.

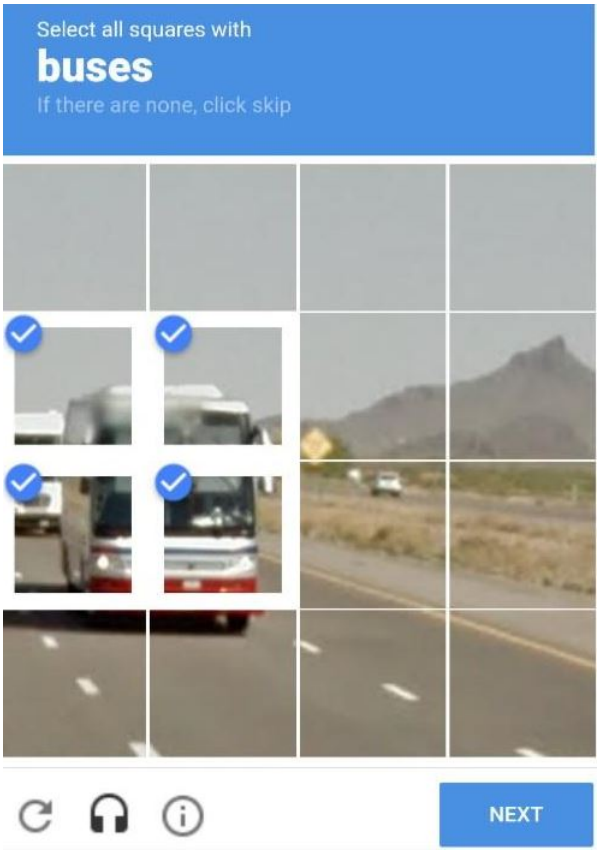
2.3 Neviditeľná CAPTCHA

Ohlásená v roku 2017, neviditeľná („invisible“) CAPTCHA vytvorená pre webové stránky je pre používateľa nepovšimnuteľná, keďže nevyžaduje žiadnu explicitnú interkaciu s testovacím systémom. Cieľom tohto CAPTCHA testu od Googlu je zbaviť sa akejkoľvek interakcie zo strany používateľov. Nemá žiadne odškŕtavacie políčko, nepožaduje prepis textu, ani spájanie obrázkov. Namiesto toho, tento druh monitoruje, ako používateľ interaguje s webovou stránkou. Programátorovi

webovej stránky stačí zahrnúť script a pridať zopár HTML atribútov k niektorým prvkom. API taktiež umožňuje explicitné vyvolanie funkcie kontroly v JavaScriptovom kóde. Kompletná dokumentácia k tomuto testu je dostupná na oficiálnej stránke[7].

3 LayerCaptcha

Mnoho obrázkových CAPTCHA testov je založených na princípe rozpoznania pôvodného znenia textu po jeho zdeformovaní, spoliehajúc sa na problematickosť počítača úlohu vyriešiť pomocou OCR. Taktiež na používateľa rozpoznať text takmer okamžite a bez problémov. To však často zlyhá úplne – nezriedkavo sa text zdeformováva natoľko, že používateľovi neostáva nič iné, než si správnu odpoveď tipnúť.



Obr. 12: Nejednoznačnosť úlohy CAPTCHA testu od Google

Iné druhy vyžadujú identifikáciu konkrétnych objektov v každej z niekoľko častí obrázka. Príkladom takéhoto overenia je napríklad dobre známa moderná CAPTCHA od Google znázornená na obrázku 12. Takéto overenie nie je bezproblémové a to hneď z dvoch hľadísk. Prvým je získavanie obrázku a následná

[7] https://developers.google.com/recaptcha/docs/invisible

automatizovaná identifikácia objektov v ňom. Napríklad na natrénovanie neurónovej siete len jedného typu objektu ako v tomto prípade autobusu je potrebná rozsiahla trénovacia sada. Ak je predmet rozpoznávania dostatočne jednoduchý na naučenie pre prevádzkovateľa webservru s CAPTCHA testom, povedzme, menšieho formátu, tak môžme počítať s tým, že je dosť jednoduchý aj pre útočníka. Ďalší problém vidno hneď na ilustrácii. Aká je správna odpoveď tohto obrázkového testu? Má používateľ vybrať aj pneumatiky? Počíta sa to ešte ako autobus? A ak áno, zahrňuje správna odpoveď aj štvorec úplne vľavo-dole, resp. je v tieni obsiahnutá aj pneumatika?

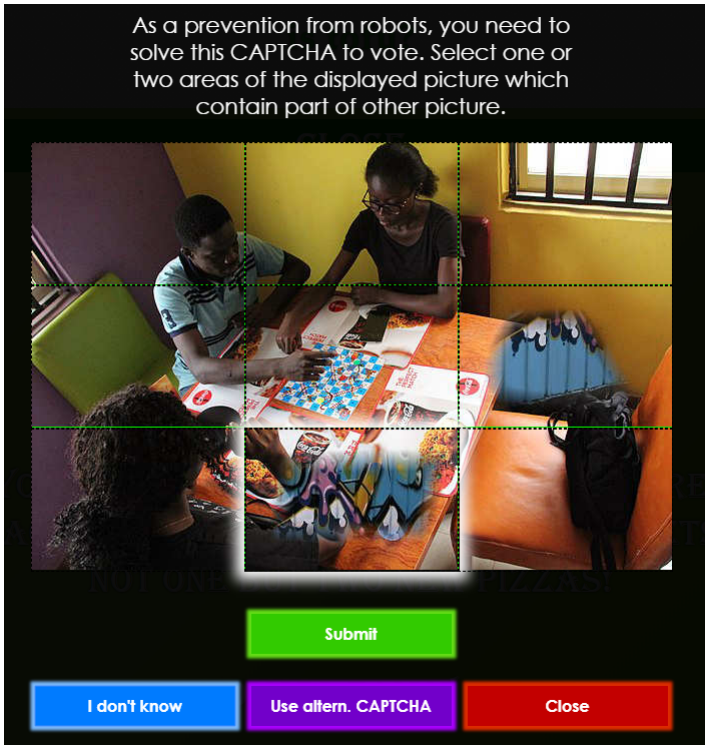
V práci navrhnutý CAPTCHA test pracuje na podobnom princípe. Stručne: obrázok bude rozdelený vždy na 9 častí a používateľ má za úlohu vybrať tie časti, ktoré obsahujú časti iného obrázka. Obrázky si CAPTCHA server (predvolene, viď sekciu 3.2.2) sťahuje z www.wikimedia.org a prekresľuje ich cez seba. Obrázky dokáže cachovať, predpripravovať, vyhľadávať optimalizované veľkosti obrázkov, meniť veľkosti a nastavovať premenlivú kvalitu JPG formátu pre dosiahnutie minimalizácie potrebného preneseného množstva informácií. Podrobnejšie v podkapitole o implementácii 3.2. Prekreslenia sú randomizované tak, aby neboli nikdy totožné, no ale aby bolo používateľovi vždy jasné, či zásah záplaty do inej častí obrázka sa považuje za správnu odpoveď alebo nie.

Funkcionalita bude predvedená napríklad na webovej stránke fiktívnej pizzérie. Po návrhu a implementácií jednoduchej HTML stránky na obrázku 13 budeme chcieť zabrániť automatizovanému hlasovaniu v malom prieskume v pravej časti. Je potreba zvoliť z niektorej možností hlasovania (možnosť je podčiarknutá podľa pozície kurzora).

Obr. 13: Fiktívna stránka s hlasovaním vyžadujúcim overenie

Voľba bola odoslaná na server, no nebola započítaná a server odpovedal s výzvou. JavaScript kód (viď sekciu 8), ktorý rozumie štruktúre výziev tohto CAPTCHA testu, po spracovaní odpovede vyvolal akciu na skripte obsluhujúcom používateľské rozhranie. Používateľovi sa otvorí modálne okno s jednoduchým a ľahko pochopiteľným minimalistickým dizajnom. Výhodou tohto modálneho

okna s veľkosťou na celú obrazovku je, že veľký obrázok CAPTCHA testu sa už návrhár stranky nemusí snažiť obtiažne zakomponovať do layoutu stránky. To by znamenalo buď nepraktické prekopanie veľkej časti stránky alebo zmenšenie obrázku na veľkosť, ktorá používateľovi nemusí stačiť.

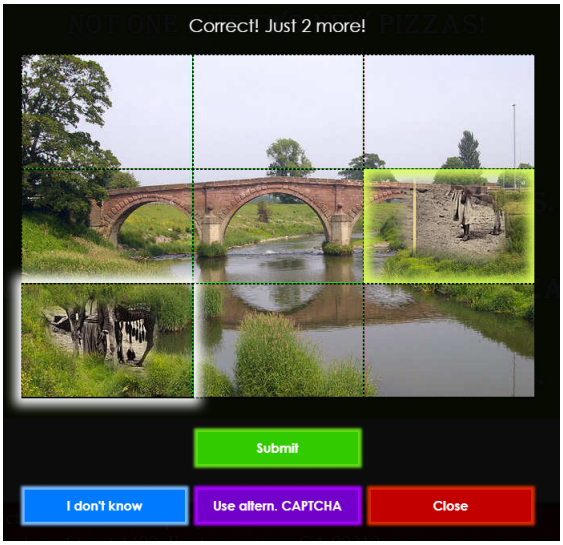


Obr. 14: LayerCaptcha GUI

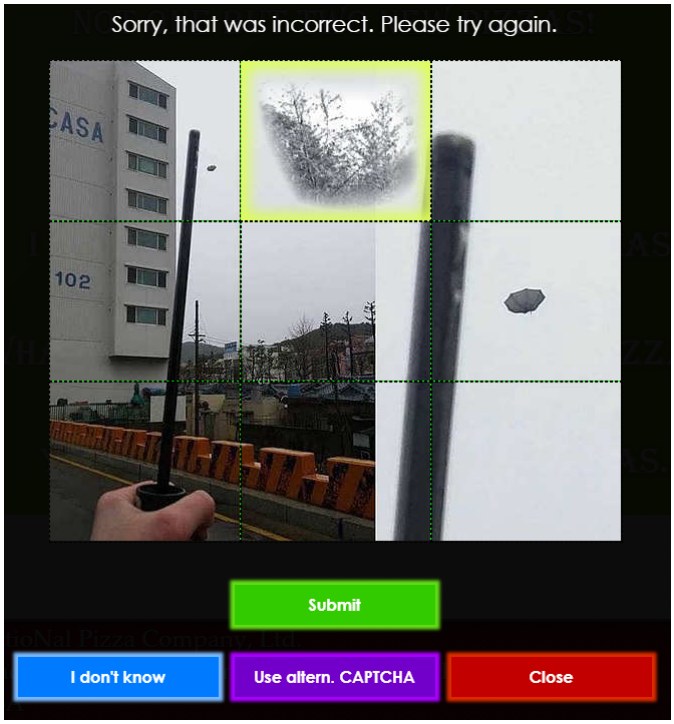
Používateľ vidí inštrukcie, čo má robiť (a prečo). Používateľ môže okno zavrieť veľkým červeným tlačidlom s nápisom Close, odoslať vyznačené odpovede alebo si opýtať nový obrázok, či použiť alternatívny CAPTCHA test. Na servri je nastavený maximálny počeť nesprávnych odpovedí. Po jeho dosiahnutí musí používateľ počkať špecifikovaný čas. Požiadavka na nový obrázok sa predvolene počíta ako polovica nesprávnej odpovedi.

Odoslala sa správna odpoveď, no správca požaduje 2 vyriešené výzvy. Skript zachytí CAPTCHA odpoveď a vie, že má GUI uvedomiť, nech zobrazí informáciu o správnej odpovedi a koľko ešte správnych odpovedí požadujeme. Pri nájazde kurzora na časť obrázka kurzor indikuje, že časť obrázka reaguje na kliky, okraje sa rozsvietia. Označené časti majú žlté okraje a sú vnútorné, aby bolo jasné, že je oblasť vybraná a už sa po nájazde kurzoru nezvýrazňujú. Na obrázku 15 vidno rozdiel medzi vyznačenou časťou (vpravo) a časťou, na ktorej stojí kurzor.

Na obrázku 16 je vidieť reakciu na informáciu od servru o nesprávnej odpovedi. Klientovi sa predvolene nepošle správna odpoveď na zle odpovedané výzvy a rovno sa zašle nová. Samozrejme s informáciou, že správne výzvu nevyriešil. Správca servru si môže zvoliť, či sa pri nesprávnej odpovedi zachová počet správnych odpovedí alebo sa vynuluje. Po zaslaní server odpovie informáciou



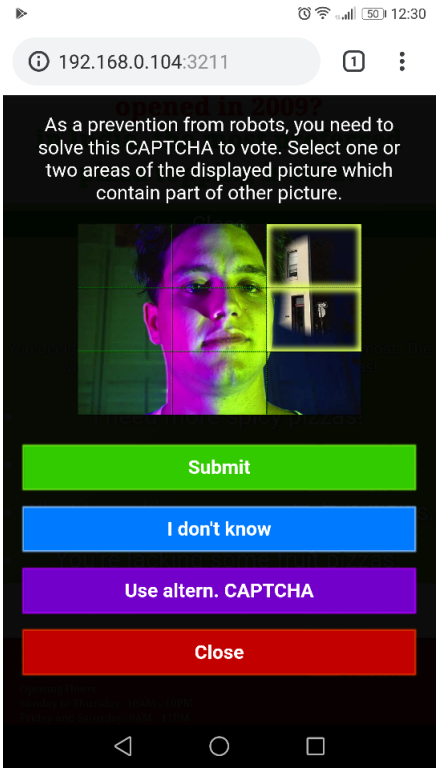
Obr. 15: Oznámenie o správnej odpovedi



Obr. 16: Oznámenie o nesprávnej odpovedi

o správnom vyriešení. Teda jedine pokiaľ to používateľovi netrvalo príliš dlho – viac ako stanovený čas na jednu odpoveď.

Používateľ je podobným spôsobom uvedomený o vypršaní časového limitu. Časový limit je len jednoduchá konštanta v obsluhujúcom PHP skripte a samozrejme môže byť zmenená. Na obrázku 17 je vidieť podobu modálneho okna pri zobrazení na portrétovo orientovaných obrazovkách, konkrétne smartfónu.



Obr. 17: GUI layout LayerCaptcha na telefóne

Grafické rozhranie modálneho okna sa prispôsobí veľkosťou prvkov, ale aj ich rozložením. Po úspešnom vyriešení úlohy sa modálne okno zavrie a používateľ už uvidí výsledky hlasovania.

3.1 Použité technológie

3.1.1 Node.js

Node.js je

- open-source
- medziplatformové
- JavaScriptové

runtime prostredie, softwarový systém navrhnutý pre programovanie vysoko škálovateľných webových aplikácií. Je to technológia umožňujúca programátorovi písať webové servre v jazyku JavaScript. Zjednocuje vývoj webových aplikácií pre jeden programovací jazyk, naproti používaniu rôznych jazykov medzi klientom a servrom. Node.js je udalosťami riadená technológia schopná asynchrónneho I/O.

Node.js vytvoril v roku 2009 Ryan Dahl ako alternatívu k Apache HTTP Servrom nezvládajúcim veľké množstvá konkurentných pripojení. JavaScript už

bol vtedy dávno dobre známy jazyk, takže rozšírenie systému Node.js po komunite webových vývojárov bolo mimoriadne rýchle. O rok neskôr bol vydaný balíčkový manažér npm – node package manager zjednodušujúci publikovanie a zdieľanie zdrojových kódov Node.js knižníc. Balíčky z npm sú prístupné cez www.npmjs.com – webovú stránku umožňujúcu vyhľadávanie balíčkov, zobrazujúcu popularitu balíčkov, dokumentácie, atď. Druhý spôsob využitia balíčkového systému je použiť nástroj príkazového riadku npm, ktorý okrem iného dokáže balíčky sťahovať, aktualizovať, editovať package.json súbory.

Node.js je stavaný na Google V8 engine implementovanom v C++. V8 kompiluje JavaScriptový zdrojový kód priamo do strojového kódu namiesto realtimeovej interpretácie. Výsledkom je pre vysokoúrovňový jazyk mimoriadna rýchlosť konkurujúca jazykom využívaných na tvorbu výkonnostno kritických aplikácií. A to aj napriek dynamickému typovaciemu systému, či automatickej správy pamäte. Preto je Node.js revolučná technológia – už si programátor nemusí vyberať medzi pohodlným vývojom a rýchlosťou, Node.js poskytuje obidva aspekty. Node.js proces nepotrebuje viacero vlákien. Narozdiel od jazykov ako C alebo Java, používa asynchrónne I/O volania, jedno vlákno na obsluhu slučky udalostí, pričom konkurentnosť medzi procesmi je spravovaná automaticky systémom. Node.js využíva dobre známe POSIX API a C knižnice, namiesto reimplementácie ich funkcionality[5].

Skripty boli testované na Node.js 8.11.3 a 10.16.0.

3.1.2 PHP

Server ukážkovej stránky beží na vstavanom PHP web servri, určenej práve na testovacie účely. Svojím vznikom predstavoval jazyk PHP akúsi webovú revolúciu. PHP je totiž napriek svojej schopnosti generovať plne dynamické stránky pomerne jednoduchý a pohodlný jazyk. PHP je multiplatformová technológia a zároveň veľmi flexibilná: okrem HTML a textových dokumentov, PHP má zabudovanú podporu pre generovanie PDF súborov, JPEG, GIF, či PNG grafiky. Výborne zvládnutá je aj podpora pre databázy. U PHP je podpora dobre známych databáz samozrejmosťou, no podporu ponúka aj pre tie menej známe[6]. Aj dnes PHP ešte tvorí backend väčšiny dynamických webových stránok. PHP 5 vydaná uprostred roka 2004 priniesla množstvo nových vylepšení. Okrem objektovo orientovaného prístupu priniesla aj nový engine, PDOs a niekoľko výkonnostných vylepšení. Dnes väčšina PHP web servrov používa verziu 7+[7]. V práci sa PHP skripty testovali na verzii PHP 5.6 a 7.2.

3.2 Implementácia

V tejto kapitole podrobnejšie vysvetlím spôsob fungovania LayerCaptcha. Algoritmus generácie, požiadavky o nový test, spracovanie servrom a následne klientom si ukážeme nahliadnutím do implementácie niektorých aspektov mechanizmu. V práci bola navrhnutá a naprogramovaná servrová časť služby generujúca

a poskytujúca LayerCaptcha testy, skript PHP triedy umožňujúci odlišovať používateľov pohodlným a objektívne orientovaným prístupom, PHP skript generujúci webovú stránku s jednoduchým hlasovacím prvkom využívajúc autentifikáciu používateľov pomocou zmienenej triedy. Ďalej bola implementovaná aj obdoba PHP web servru Node.js Express.js[8] aplikácia. Ďalej samotnú webovú stránku okrem HTML a CSS súborov vyžívajúcu tiež scripty na komunikáciu s PHP servrom implementujúcim logiku hlasovania, skript na spracovanie CAPTCHA súvisiacích odpovedí a generujúcu grafické používateľské rozhranie, s ktorým používateľ interaguje pri riešení výzvy.

3.2.1 Odpovedanie na požiadavky

Program generujúci CAPTCHA testy a posytkujúci ich záujemcom je servrová Node.js aplikácia. Spustenie aplikácie cez hlavný skript „app“ spustí HTTP server počúvajúci na adrese a porte daných v nastaviteľných konštantách, forkne proces na poskytovanie statických súborov a započne beh manažéra aplikácie. Manažér „Mgmt“ spravuje beh celého procesu generovania, cachovania, poskytovania výziev, kontrolovania potreby generácie. Jeho funkcionalitu, tak ako aj všetky nastaviteľné konštánty si ukážeme neskôr. Odpovedanie prebieha nasledovným spôsobom: po tom, čo nám manažér predá hotový CAPTCHA test objekt `captcha`, pripravíme hlavičku HTTP odpovede indikujúcu, že typ odpovede bude JSON objekt(application/json podľa IANA špecifikácie). Akýkoľvek typ cachovania odpovedí je pre nás v tomto prípade nežiadúce. S každou novou prosbou o CAPTCHA test by mala prísť nová, jedinečná a predtým nevygenerovaná CAPTCHA. Preto v hlavičke nastavíme `Cache-Control` na hodnotu `no-store`. Odpoveď pozostáva z reťazca obsahujúci JSON reprezentáciu finálnej podoby testu – teda správnu odpoveď a odkaz na staticky poskytovaný obrázok, ako vidno v kóde 3. Manažér má testy už predpripravené a ak počas generácie vznikne chyba, manažér ju len zaloguje a vygeneruje test znovu. Keďže beh manažéra je nekonečná slučka implementovaná ako časový interval, aplikácia bude reagovať na signály prerušenia ako SIGINT uvedomením manažéra, ktorý po prijatí signálu ukončí svoju prácu.

```
1 {"path":"/c1.jpg","answer":"35"}
```

Zdrojový kód 3: Forma odpovede

`Mgmt` je trieda manažujúca chod aplikácie. Presnejšie, je to trieda len, dajme tomu, syntakticky. JavaScript, narozdiel od klasických objektovo orientovaných jazykov triedy nemá. Slovo class v jazyku JavaScript (od ES6) je pokus dodať programátorom JavaScriptu zvyknutí na jazyky ako Java typický rys týchto programovacích jazykov – triedy. V skutočnosti JavaScriptové triedy sú si so

[8] viac o Express.js v 4.1.2

skutočnými triedami podobné do určitej miery len syntakticky. V JS má každý objekt vlastnosť prototype, čo je len referencia na iný objekt[8]. Pri volaní s „new“ X sa vytvorí objekt, ktorého prototyp je práve X. Trieda objektov sú v skutočnosti objekty, ktorých prototype vlastnosť referencuje ten istý objekt. Ak objekt nemá vlastnosť, ku ktorej na ňom pristupujeme (možná kontrola pomocou metódy `hasOwnProperty`), pokračuje sa v hľadaní v prototypovom reťazci, až kým sa vlastnosť nenájde alebo prototypový nekončí. V JavaScripte sa teda kopíruje referencia na delegovaný objekt – preto sa o JavaScripte zvykne hovoriť ako o prototypovom OOP, než klasickom triedovom.

Keďže manažér alebo jeho obdoby budú použité na predpripravonie testov v celej práci, popíšeme si teraz, čo má na starosti. `Mgmt` si bude udžiavať rad hotových CAPTCHA testov pripravených na odoslanie. Zvlášť si bude udržiavať už zoznam odoslaných testov. Manažér periodicky maže staré odoslané testy – nemá zmysel si udržiavať obrázky dlhšiu dobu, používateľovi trvá jedno vyriešenie len niekoľko sekúnd. CAPTCHA sa zmaže z tohoto zoznamu, ale aj súbor na disku obsahujúci telo obrázka. Dôležitý aspekt mechanizmu generovania testov je cachovanie stiahnutých obrázkov. Obrázky, ktoré si z internetu aplikácia zaobstará, hneď nezahodí, ale odloží si ich na neskôr a znovupoužije. Každý obrázok je reprezentovaný ako objekt triedy `LCImage`, ktorý ma zopár jednoduchých metód a stavových premenných. Aký má vlastne význam čakať pred pridaním do cache? Uvedomme si, že ak by boli obrázky ihneď znovupoužiteľné, útočník by si mohol poprosiť o nový CAPTCHA test, ktorý by mohol dostať z obrázkov, z ktorých bol vytvorený pôvodný. Ani z dvoch rovnakých obrázkov sa nikdy nevygeneruje rovnaký test (detaily generovania budú ukázané neskôr), no pri väčšom šťastí by sa mu mohlo podariť nájsť a zneužiť podobné črty obrázkov a použiť to na prelomenie pôvodného testu.

Na požiadavky sa teda odpovedá predpripravenými testami. Ak sa pri odpovedaní zistí, že počet predpripravených testov je príliš nízky, zvýši sa kapacita. Periodická kontrola sa o doplnenie kapacity postará. Ak sa očakáva odpoveď, ale nemáme žiaden test pripravený na odoslanie, požiadavka sa pridá do rady čakateľov a odpovedá sa hneď po dokončení generovania nového testu. Periodická kontrola pri odovzdaní hotového testu manažérovi vždy najprv uspokojí čakateľov. Až keď žiadni čakatelia nie sú, pridáva sa do zoznamu pripravených testov. Požiadavkám sa teda odpovedá spôsobom First Come, First Served.

3.2.2 Sťahovanie obrázkov

Ak `Mgmt` pri periodickej kontrole zistí, že počet pripravených testov nedosahuje hodnotu aktuálne potrebnej kapacity, zaobstará si 2 obrázky, z ktorých vytvorí nový CAPTCHA test. Obrázky sa primárne snaží znovupoužiť, no ak sa to nepodarí, poprosí o obrázok `imageProvidera`. To je konštanta `Mgmt` modulu, v ktorej je predvolene importovaný v práci implementovaný modul na získavanie obrázkov, zvaný `WikiImageProvider`. Používateľovi programu nič nebráni použiť vlastný modul poskytujúci obrázky. Zdrojový kód 4 predstavuje metódu `getImg` nášho poskytovateľa obrázkov `WikiImageProvider`. Keďže takmer každá metóda

sa bude vykonávať asynchrónne, použijem štýl zápisu **`async/await`**. V JavaScripte predstavuje async/await[9] mimoriadne pohodlný spôsob[9], ako pracovať s asynchronnými funkciami. Po tom, čo je metóda označená kľúčovým slovom **`async`**, môže v sebe využívať **`await`** – čakanie na výsledok Promisu nachádzajúcim sa za **`await`**, namiesto starého známeho `.then()` a **`.catch`**`()` vedúcemu k objemnému kódu. Ako vidno z kódu 4, princíp je nasledovný:

1. `_getImgLink` prejde HTML dokument obaľujúci náhodný obrázok (ponúkané cez wikimedia API[10]. To zahrňuje získanie optimálnej veľkosti obrázka. Keďže Wiki API to predvolene jednoducho neumožňuje, ostáva na to aplikácia sama. Optimálna veľkosť obrázka je určená v konštantách. Metóda nájde veľkosť najpodobnejšiu optimálnej. Optimálna veľkosť obrázka by nemala presahovať rozmer 700x700 pixelov – človek na vyriešenie CAPTCHA testu nepotrebuje vidieť obrázok v 4K UHD rozlíšení. Metóda taktiež vyhodí výnimku, ak je obrázok príliš malý. Minimálna veľkosť obrázka je znovu definovaná v konštanách. `Mgmt` čakajúce na obrázok pochopí vyhodenie výnimky, zaloguje dôvod a pokúsi sa stiahnúť obrázok odznova.

2. `getExtFromImgLink` vráti formát, presnejšie príponu obrázka. Návratová hodnota môže byť len reťazec z jpg, png a **`null`** pre všetko ostatné. Ak je teda formát nepodporovaný, uvedomí sa o tom `Mgmt` vyhodením chyby.

3. `_downloadImgData` stiahne telo obrázku pomocou Node modulu `request-promise-native`

4. `_arrangeTempFile` vytvorí nový dočasný súbor pomocou Node modulu `tmp-promise`. Prefix je nastaviteľný v konštantách a je predvolene „LC-TMPF-“. Zvyšok názvu zaistí modul a prípona je prípona zistená v druhom kroku. Modul zaistí vytvorenie súboru v adresári systémovo určenom na dočasné súbory. V mojom prípade bola teda plná cesta k dočasnému súboru „C:\Users\Kristian\AppData\Local\Temp\LC-TMPF-3752mTpF3D9WS95o.jpg“. Objekt vrátený touto metódou bude obsahovať cestu k súboru a taktiež funkciu na jeho zmazanie.

5. `_writeAndGetImg` zapíše obrázok do dočasného súboru a vráti nový `LCImage` obsahujúci cestu k súboru a funkciu na jeho zmazanie.

`Mgmt` teda dostane jeden `LCImage`, na výrobu CAPTCHA testu sú ale potrebné 2, preto tento postup spustí hneď dvakrát[11]. Vykonávanie týchto dvoch požiadaviek sa môže prekrývať, no nie je v žiadnom prípade paralelné. Nezabúdajme, že Node.js proces je jednovláknový. To ale vôbec nevadí: kým prvé `getImg`

[9] od ECMAScript 2016
[10] https://commons.wikimedia.org/wiki/Special:Random/File
[11] ak nie je prístupný obrázok z cache

napríklad čaká na request odpoveď, aby mohla spracovať spomínaný HTML dokument, druhé `getImg`, ak mu to slučka udalostí umožní, môže začať s vykonávaním kódu a neprestane, až kým nenarazí na ďalšie asynchrónne čakanie (alebo kým funkcia nedobeží dokonca).

```
1 const getImg = async () => {
2    const imgLink = await _getImgLink(),
3       ext = WC.getExtFromImgLink(imgLink);
4    if (!ext)
5       return Promise.reject(new Error(`Unsupported file extension of
            link ${imgLink}`));
6    const imgData = _downloadImgData(imgLink), tmpFileObj =
       _arrangeTempFile(ext);
7    return await _writeAndGetImg((await imgData).data, await
       tmpFileObj);
8 };
```

Zdrojový kód 4: Postup sťahovania obrázka

3.2.3 Generovanie tvaru

Po tom, čo máme po ruke 2 obrázky, z ktorých budeme generovať CAPTCHA test obrázok, môžme ich predať ako argument funkcie `createFromPair`, čo je funkcia modulu `LCCreator`. Funkcia berie ako argumenty pole (dvoch) `LCImage` obrázkov a funkciu generujúcu vzorku, podľa ktorej bude jeden obrázok vkreslený do druhého. Vzorka je teda tvar časti vkresleného obrázka. V práci som navrhol a implementoval modul generujúci takúto vzorku, `PatternProvider`. Na nasledujúcich stranách sa budem venovať mechanizmu generácie vzorky, keďže práve tvar prekresleného obrázka určuje, či bude naša CAPTCHA jednoducho prelomiteľná pomocou rôznych techník na detekciu hrán a identifikáciu tvarov. Funkcia generujúca vzorku je okrem šírky a výšky volaná ešte s dvoma silami rozvratu hraníc vzorky – tie určujú, ako veľmi sa budú randomizovať vrcholy pomyselného štvoruholníku vzorky a riadiace body kriviek. Každá vzorka bude určená vrcholmi, medzi ktorými vytvoríme 4 krivky. „Padding“ vzorky je vždy štvrtina, pätina alebo šestina (určené náhodne) šírky/výšky. Určuje to, ako ďaleko od okraja sa majú nachádzať vrcholy vzorky. Takto to ale nenecháme, hneď po vytvorení objektu s vrcholmi ho predáme metóde `disruptPatternBounds`, ktorá vrcholy náhodne rozmiestni. Neprekročí ale padding, teda vrchol nepremiestni mimo hraníce. Akceptuje tiež parameter určujúci akou silou majú byť vrcholy rozhádzané. 0 reprezentuje nijak, 0.5 maximálne do polovice paddingu, 1 maximálne do hodnoty paddingu. Bez udania tohto parametru je parameter desatina z náhodného celého čísla od 0 po 10 (vrátane 0 a 10), vygenerované pre každý vrchol znova. Smer, teda vľavo/vpravo a hore/dole rozhodenia pozície vrcholu je vždy určený náhodne s rovnakou pravdepodobnosťou. 4 krivky budeme

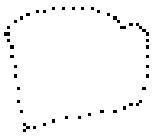
generovať pomocou algoritmu de Casteljau na generáciu Bézierovych kriviek, založeného na opakovanej lineárnej interpolácii[10]. Bézierova krivka stupňa n je určená riadiacím polygónom ako lámaná čiara určená úsečkami medzi kontrolnými bodmi $P_0, \ldots, P_n$. Pre riadiací polygón s $n+1$ bodmi potrebujeme vykonať n krokov algoritmu, aby sme získali bod krivky. Parametrizácia Bézierovej krivky stupňa n je daná vzťahom,

$$P(t) = \sum_{i=0}^{n} \binom{n}{i} (1-t)^{(n-i)} t^i P_i, t \in \langle 0, 1 \rangle$$

Teda v našom prípade kubická Bézierova krivka sa dá vyjadriť vzťahom

$$P(t) = \sum_{i=0}^{3} \binom{3}{i} (1-t)^{(3-i)} t^i P_i = (1-t)^3 P_0 + 3(1-t)^2 t P_1 + 3(1-t) t^2 P_2 + t^3 P_3$$

Pre každú hranu pomyselného štvoruholníka sa vygenerujú riadiace body kubickej beziérovej krivky. Bude ich spolu 4 ($n = 3$), z toho počiatočný a koncový bod sú vrcholy pomyselného štvoruholníka. Ostáva teda generovať 2 body. Vygenerujú sa tak, aby pomyselnú úsečku medzi koncovými bodmi rozdeľovali na tretiny horizontálne i vertikálne. Následne neokrajové kontrolné body rozhádžeme podobným spôsobom ako pri vrcholoch pomyselného štvoruholníka. Nakoniec už vygenerujeme Bézierovu krivku vzťahom uvedeným vyššie. Krok t si zvolíme ako tretinu z $d(P_0, P_1) + d(P_1, P_2) + d(P_2, P_3)$, kde d je euklidovská vzdialenosť dvoch bodov. Takto zvolené krokovanie síce vygeneruje krivko rýchlo, no produkuje nespojité krivky, ako je vidieť na obr 18.

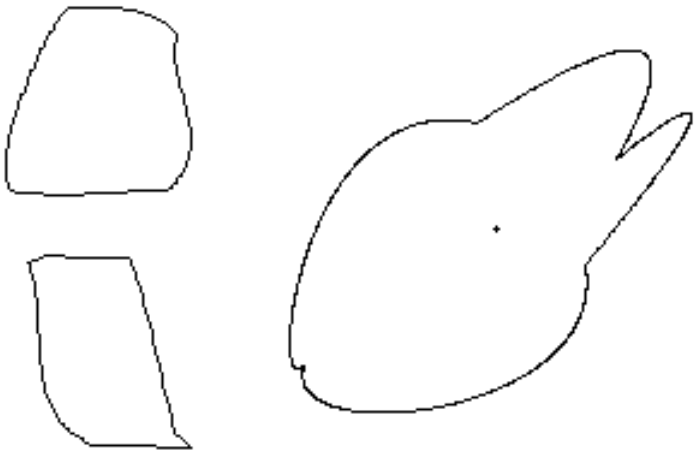


Obr. 18: Výsledok riedkeho krokovania pri generovaní Bézierovej krivky

Po každom zaznačení pixelu teda spojím pixel s predošlým zaznačeným. Dva body spojím jednoduchým algoritmom:

```
1: x ← x_0
2: y ← y_0
3: d_x ← x − x_1
4: d_y ← y − y_1
5: while |d_x| > 1 or |d_y| > 1 do
6:     if |d_x| > |d_y| then
7:         if d_x < 0 then
8:             x ← x + 1
9:         else
10:            x ← x − 1
11:        end if
12:    else
13:        if d_y < 0 then
14:            y ← y − 1
15:        else
16:            y ← y + 1
17:        end if
18:    end if
19:    d_x ← x − x_0
20:    d_y ← y − y_0
21: end while
```

Presúvam sa tým smerom, ktorým som ku koncovému bodu vzdialenejší, po ceste zaznačujem pixely. Príklady výsledných tvarov môžme vidieť na obrázku 19. Dve vzorky vľavo o rozmeroch 100x100 a vzorka vpravo o veľkosti 200x200 pripomínajúca zajaca[12] so zaznačeným stredom pomyselného štvoruholníka.

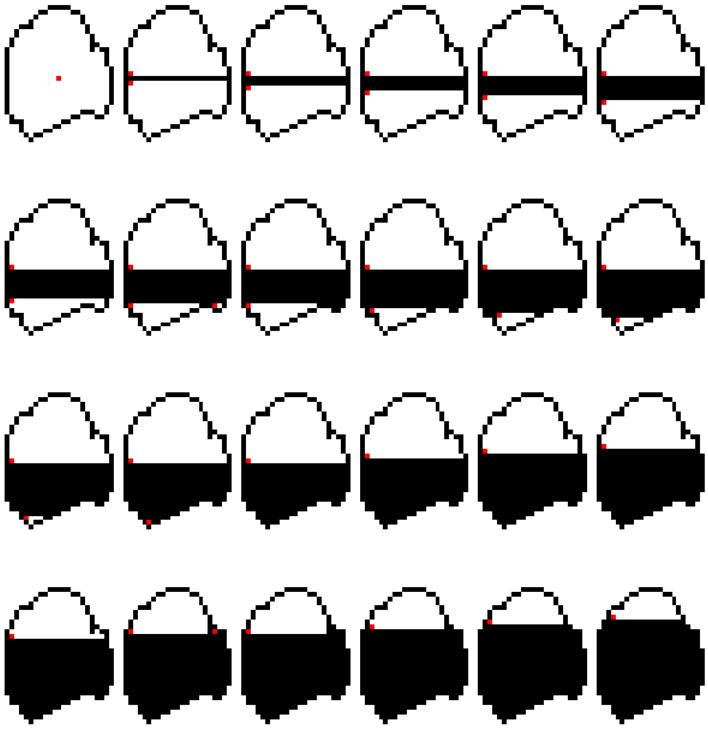


Obr. 19: Príklady vygenerovaných vzoriek

Po tom, čo sú hotové obrysy vzorky, sa môžem pustiť do jej vypĺňania. Vypĺňam scanline fill algoritmom. Zo začiatku sa umiestní do stredu pomyselného

[12]alebo kačku! https://www.google.com/search?q=rabbit+duck+illusion&tbm=isch

štvoruholníka seed, ktorý bude slúžiť ako počiatok. Vyplní sa riadok, na ktorom sa nachádza seed a zapamätá sa rozsah vyplnenej čiary a to súradnice x_L, x_R - teda najmenšiu a najväčšiu x-ovú hodnotu. Po vyplnení čiary sa prejde cez všetky pixely od $[x_L, y-1]$ po $[x_R, y-1]$ a rozmiestna sa nové seeds všade, kde sa narazí na začiatok nevyplnenej oblasti – pridajú sa na vrch zásobníka. Analogicky pre $[x_L, y+1]$ až $[x_R, y+1]$. Ďalej berieme z vrchu zásobníka a opakujeme, kým nie je prázdny. Ilustrácia (neúplného) jednoduchého vypĺňania vzorky o veľkosti 25x40 je na obrázku 20. Červeným sú znázornené seeds.

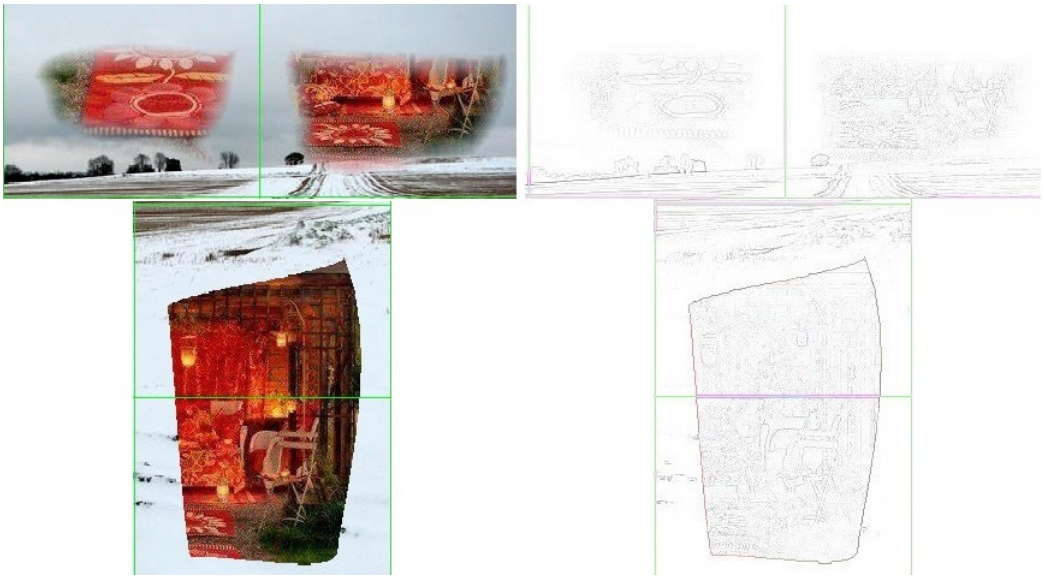


Obr. 20: Vypĺňanie vzorky algoritmom scanline fill

Posledný krok je vyhladenie hrán – pridanie transparencie hranám. Transparencia sa pridá krajným h pixelom vzorky, kde pw je šírka vzorky, ph je jej výška a $h = 1 + \sqrt{3 \min(pw, ph)}$, následne sa h zaokrúhli. Prejde sa cez všetky pixely okrajov vzorky, vypočítajú sa ich transparencie vzťahom i/h, kde i je vzdialenosť aktuálneho pixelu od krajného pixelu +1. Na (de)kódovanie obrázkov, zmenu JPG kvality a (ak je v konštantách nastavené) na zmenu veľkosti pôvodných rozmerov obrázka som použil modul `imagejs`[13]. Porovnajme si teraz 2 časti CAPTCHA obrázkov vytvorené z rovnakých predloh, ale pri vytvorení jednej hrany neboli vyhladené. Následne na tieto časti aplikujem techniku detekcie hrán pomocou Laplaciána. Všimnime si, že technika vyhladenia hrán značne zťažila detekciu našej vzorky.

Modul vytvárajúci už hotové obrázky `LCCreator` sa potrebuje rozhodnúť,

[13] https://github.com/guyonroche/imagejs



Obr. 21: Obrázok bez (dole) a s (hore) vyhladením po detekcii hrán

koľko vzoriek má generovať. Rozhoduje sa rovnomerne medzi jednou, dvoma a jednou veľkou (zaberajúca 2 miesta v mriežke). Ak jedna veľká, tak sa opäť náhodne rozhodne, či má byť prekreslená cez dve oblasti horizontálne alebo vertikálne. Po prekreslení, v závislosti od nastaviteľných preferencií, môže obrázku zmeniť veľkosť a následne ho uloží ako vo formáte JPG s kvalitou `CONSTS.jpgQuality`. Hotový `LCaptcha` objekt predá volajúcemu `Mgmt`, ktoré ho poskytne čakateľom na CAPTCHA test alebo ho vloží medzi ostatné predpripravené testy, ak žiadni čakatelia nie sú. Odoslaný obsah obsahuje JSON reprezentáciu cesty k súboru obrázku a správnu odpoveď. Obrázky ale nebudeme poskytovať týmto procesom. Uvedomme si, že Node.js proces je jednovláknový a poskytovanie objemných statických súborov môže zbytočne brzdiť proces generovania a sťahovania. Preto vytvoríme pri štarte aplikácie ešte jeden Node.js proces. Ten už môže byť spustený hostujúcim systémom v novom vlákne. Narozdiel ale od POSIX volania funkcie `fork` napríklad v jazyku C, fork nevytvorí nový proces ako klon aktuálneho.

3.2.4 Nastaviteľné konštanty

Pre Node.js proces generácie sa dajú nastaviť globálne konštanty ovplyvňujúce tvorbu CAPTCHA testov. Konštanty sa dajú nastaviť klasickým spôsobom ako argumenty spusteného procesu formou „-názovParametru hodnotaParametru“. Parametre nemusia byť zadané žiadne. Ukážme si teraz zoznam nastaviteľných konštánt a ich predvolené hodnoty, ktoré vstupujú do platnosti, ak nie sú predefinované argumentami procesu.

`checkInterval`

Ako často sa bude kontrolovať počet predpripravených testov a následne

spustené generovanie, ak sa zistí, že je počet hotových výziev nedostatočný. Doba je v milisekundách. Predvolene 3000.

captchaSaveDirectoryPath
Cesta k adresáru, kde sa budú ukladať obrázky hotových výziev. Predvolene ’./captchas’.

capacityCutbackInterval
Udáva, ako často sa zníži minimálna požadovaná kapacita predpripravených testov. Doba je v milisekundách. Predvolene $1000 \cdot 60 \cdot 2$

capacityCutbackMinPercentage
Udáva minimálnu hodnotu podielu hotových testov a kapacity, ktorá musí byť splnená, aby došlo k zníženiu kapacity. Predvolene $0,9$.

capacityDynamic
Ak je nastavené na hodnotu nepravdy, nutný počet predpripravených testov bude pri celom behu programu rovnaký a rovný konštante počiatočnej kapacity. Predvolene true.

initialCapacity
Počiatočná kapacita. Predvolene 3.

ptrnBoundsDisruptionSeverity
Udáva silu randomizácie pozície vrcholov pomyselného štvoruholníka vzorky ako racionálne číslo z intervalu 0 až 1. 0 znamená nerandomizovať vôbec, 1 znamená maximálne ku okraju hraníc vzorky. Predvolene 1.

ptrnBoundsDisruptionSeverity
Udáva silu randomizácie pozície riadiacich bodov štyroch kriviek z intervalu 0 až 1. Hodnoty majú rovnaký význam ako pri predchádzajúcej konštante. Nedefinovaná alebo prázdna konštanta znamená, že pre každý bod náhodne určíme silu ako racionálne číslo v rozmedzí intervalu 0 až 1. Predvolene 1.

ptrnRepeatAttemptDelay
Minimálna doba odkladu generácie vzorky po zlyhaní generácie.[14] Doba je v milisekundách. Predvolene 50.

ptrnVisualizeCreation
Či má vizualizovať postup generácie vzorky. Vizualizácia vytvorí niekoľko desiatok obrázkov do adresára určeným konštantou nižšie. Ide len o „debugovaciu“ funkcionalitu, pri nasadení by mala byť vypnutá. Predvolene false.

[14] povedzme, že je to len „bezpečnostné opatrenie“. Nie je mi známa žiadna chyba v algoritme, kvôli ktorému by generovanie vzorky mohlo zlyhať

ptrnVisualizePath
: Adresár, kde sa obrázky vizualizácie budú generovať. Predvolene je to adresár 'public/images'.

ptrnSmoothen
: Určuje, či sa majú vzorky po hranách vyhladzovať transparenciou. Predvolene true.

optimizeSize
: Určuje, či sa majú pri sťahovaní obrázkov preferovať rozmery uvedené konštantami nižšie. Predvolene true.

optimizeResizeWidth
: Preferovaná šírka obrázku v pixeloch. Predvolene 640.

optimizeResizeHeigth
: Preferovaná výška obrázku. Predvolene 480.

forceResize
: Určuje, či sa má výsledný CAPTCHA obrázok preškálovať na optimálnu veľkosť. Predvolene false.

resizeAlgorithm
: Ak sa má meniť veľkosť obrázka, tak určuje algoritmus zmeny veľkostí. Podľa dokumentácie cudzieho modulu sú povolené voľby reťazce nearestNeighbor, bilinearInterpolation, bicubicInterpolation, hermiteInterpolation, bezierInterpolation. Predvolene nearestNeighbor.

tmpPrefix
: Prefix dočasného súboru. Predvolene 'LC-TMPF-'.

imgRepeatAttemptDelay
: Minimálna doba odkladu sťahovania obrázka po tom, čo je zahodený napríklad kvôli nedostatočnej veľkosti alebo nepodporovaného formátu. Doba je v milisekundách. Predvolene 200.

imgMinWidth
: Minimálna šírka obrázka na to, aby sa mohol použiť ako obrázok testu. Predvolene 250.

imgMinHeigth
: Minimálna výška obrázka na to, aby sa mohol použiť ako obrázok testu. Predvolene 250.

jpgQuality
: Kvalita obrázkov formátu JPG, ktoré bude program generovať. Hodnoty 1 až 100. Predvolene 55.

cachingEnabled
: Určuje, či môžu byť stiahnuté obrázky znovupoužité. Predvolene true.

cacheDelayBeforeReuse
: Určuje, po akej dobe môže byť obrázok znovupoužitý. Doba je v milisekundách. Predvolene $1000 \cdot 60 \cdot 2$.

cacheMaxAdditionalDelay
: Určuje strop časového odkladu na znovupoužitie. Doba je v milisekundách. Doba pridania obrázku do zoznamu znovupoužiteľných je náhodné číslo medzi X a $X + Y$, kde X je predchádzajúca konštanta, a Y je táto. Predvolene $1000 \cdot 60 \cdot 2$.

cacheReuseLimit
: Určuje maximálny počet znovupoužití obrázka. Predvolene 3.

cleanupOnExit
: Určuje, či má program pri štarte zmazať obrázky v `captchaSaveDirectoryPath`. Predvolene true.

cleanupOnStart
: Určuje, či má program na konci zmazať obrázky v `captchaSaveDirectoryPath`. Predvolene true.

captchaMaxAge
: Určuje, po akom čase má program zmazať vygenerovaný CAPTCHA obrázok. Doba je v milisekundách. Predvolene $1000 \cdot 60 \cdot 15$

serverPort
: Určuje port HTTP servru, ktorý odpoveda na požiadavky o výzvu. Predvolene 4000.

staticServerPort
: Určuje port HTTP servru, ktorý poskytuje obrázky výziev. Predvolene 4001.

staticFileCache
: Určuje doporučenú dobu uchovania stiahnutého obrázku. Doba je v sekundách. Predvolene 600.

3.3 Robustnosť LayerCaptcha

Nech je správna odpoveď práve dvojica z 9 oblastí. Šanca uhádnutia pri náhodnom volení možností je nasledovná: 2 správne možnosti z 9, teda $\frac{2}{9}$; ďalej šanca, že uhádneme druhú možnosť je 1 správna možnosť z 8, teda $\frac{1}{8}$, spolu teda $\frac{2}{9} \cdot \frac{1}{8} \doteq 2,78\%$ Ako sme videli v predošlej sekcii o implementácii, je rovnaká šanca, že sa prekreslenie odohraje podľa jednej z troch možností:

1. jedno prekreslenie v jednej oblasti

2. jedno prekreslenie cez 2 oblasti

3. 2 prekreslenia v dvoch rôznych oblastiach

Šanca uhádnutia avšak počítala stále s dvoma možnosťami, teda v tretine sa budeme mýliť. $\frac{2 \cdot 1 \cdot 2}{9 \cdot 8 \cdot 3} \doteq 1,85\%$ je teda šanca uhádnutia pri úplne náhodnom hádaní a to vždy dvojíc. Všimnime si ale: šanca, že správne odpovede spolu budú susediť[15], nie je 1/3, ale vyššia, keďže 2 prekreslenia miesto jedného prekreslenia cez 2 oblasti nevylučujú možnosť, že spolu budú susediť. Inými slovami, možnosti 2 a 3 môžu generovať rovnaké správne odpovede. Nech nastane možnosť generácie 3. Je $9 \cdot 8 = 72$ možností, ako dvojice vybrať. Ktoré z nich sú ale susedné? V prvom riadku 3x3 mriežky, ak vyberieme prvú oblasť, dostávame 2 rôzne možnosti, ktorými môžme vybrať ďalšiu oblasť tak, aby susedila s už vybranou – a to oblasť pod ňou a vpravo vedľa nej. Ak ako prvú oblasť vyberieme v 3x3 mriežke prostrednú z prvého riadku, existujú 3 rôzne možnosti, ktorými môžme vybrať ďalšiu oblasť tak, aby susedila už s vybranou – a to oblasť pod ňou, vpravo vedľa nej a vľavo od nej. Prípad vľavo od nej už máme ale započítaný v predošlej úvahe a na poradí nezáleží, preto tento nezapočítame. Zatiaľ teda vieme, že 4 možnosti zo 72 sú susediace. Tretia oblasť v prvom riadku 3x3 mriežky dáva už len jednu druhú možnosť – oblasť pod ňou. Uvedomme si, že pre posledný riadok mriežky platia podobné úvahy, ako pre prvý, no s tým rozdielom, že budeme započítavať oblasti nad, a nie pod prvými. Spolu je to zatiaľ $5 + 5$. Do stredného už sme započítali susedov pri úvahach o prvom a poslednom riadku, ostávajú už len horizontálne susedstvá. Prvá s druhou, prvá s treťou a druhá s treťou. Vieme teda, že ak umiestňujeme dve vzorky podľa možnosti 3, je vlastne $13/72 \doteq 18,06\%$ šanca, že umiestnime dve správne odpovede vedľa seba. Skombinujme úvahu s možnosťou číslo 2. V mriežke 3x3 je 12 možností, ako do nej umiestniť vzorku 1x2, resp. 2x1. Ak teda budeme hádať vždy len dvojice čísel indexov, ktoré spolu v mriežke 3x3 susedia, dostávame šancu $1/12 \doteq 8,33\%$ za predpokladu, že vždy nastáva možnosť 2. Možnosť 2 ale nastáva iba v jednom z troch prípadov, teda šanca bude tretinová, čo je približne $2,78\%$. K tomu máme ale ešte tretinovú šancu, že nastane možnosť 3, kde šanca, že sa umiestnia vzorky susedne je 13/72 a že uhádneme práve tu správnu je $\frac{13 \cdot 1}{72 \cdot 12} \doteq 1,51\%$. Najvýhodnejšie je teda pre útočníka hádať vždy susedné oblasti, čo teda predstavuje $2,78 + 1,51\frac{1}{3} \doteq 3,28\%$ šancu na uhádnutie jediného obrázka. Šanca uhádnutia je teda veľmi nízka, pri dvoch alebo viac obrázkoch extrémne nepravdepodobná. LayerCaptcha je teda z pohľadu náhodného hádania dostatočne robustná.

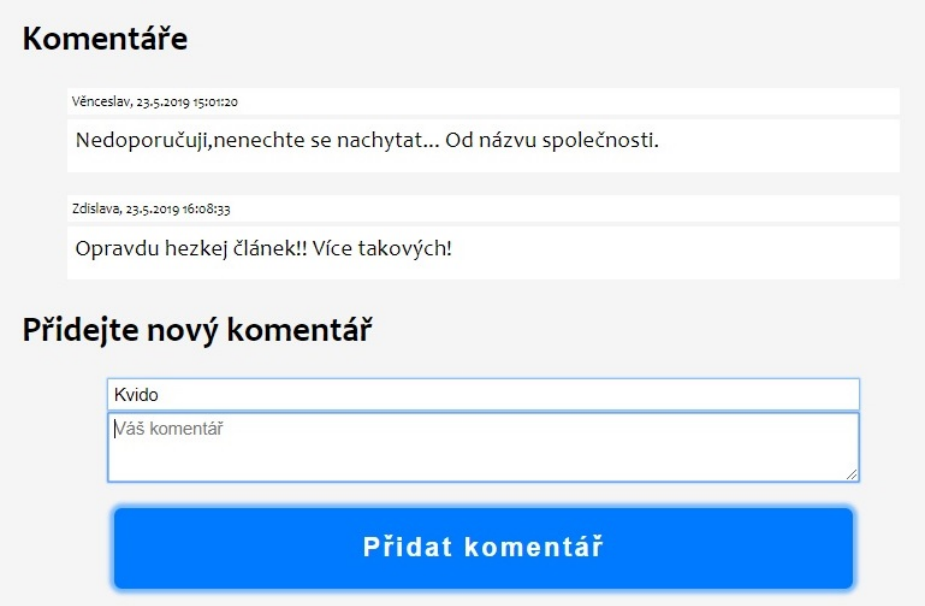
Inak to nie je ani v prípade pouzívania správnych odpovedí. Tvary vzoriek sú generované náhodne, umiestňované náhodne, o náhodných rozmerov. Obrázky použité na CAPTCHA testy sú predvolene sťahované z commons.wikimedia.org, ktoré disponuje niekoľko desiatkám miliónom obrázkov. CAPTCHA je teda z hľadiska používania známych odpovedí dostatočne robustná.

[15] štvorsmerne

Ako to býva u obrázkových CAPTCHA testov zvykom, z automatizovaných útokov je strojové učenie aj tu najslabším článkom. Ak je kód LayerCaptcha verejný, tak má útočník pre svoju neurónovú sieť k dispozícii neobmedzené množstvo dát. S výzvou totiž posielame aj správnu odpoveď. Samozrejme, existuje jednoduchší spôsob. Útočník nemusí stárnuť pri trénovaní neurónovej siete. Generovanie transparentných hraníc vzorky nie je nijak randomizované. Hlavne v rohoch vzorky, pri vrcholoch pomyselného štvoruholníka sa dajú nájsť pri niektorých obrázkoch isté podobnosti. Ak by sa podarilo tieto podobnosti útočníkovi jednoznačne podchytiť, dokázal by pomocou techník analýzy a spracovania obrazu detegovať oblasti so vzorkou. CAPTCHA je teda v tomto smere menej robustná, než pri predchádazjúcich dvoch hľadiskách. Stále však disponuje dostatočnej robustnosti – natrénovanie neurónovej siete nie je rýchly proces a detekcia zmienených podobností by nemala byť triviálna.

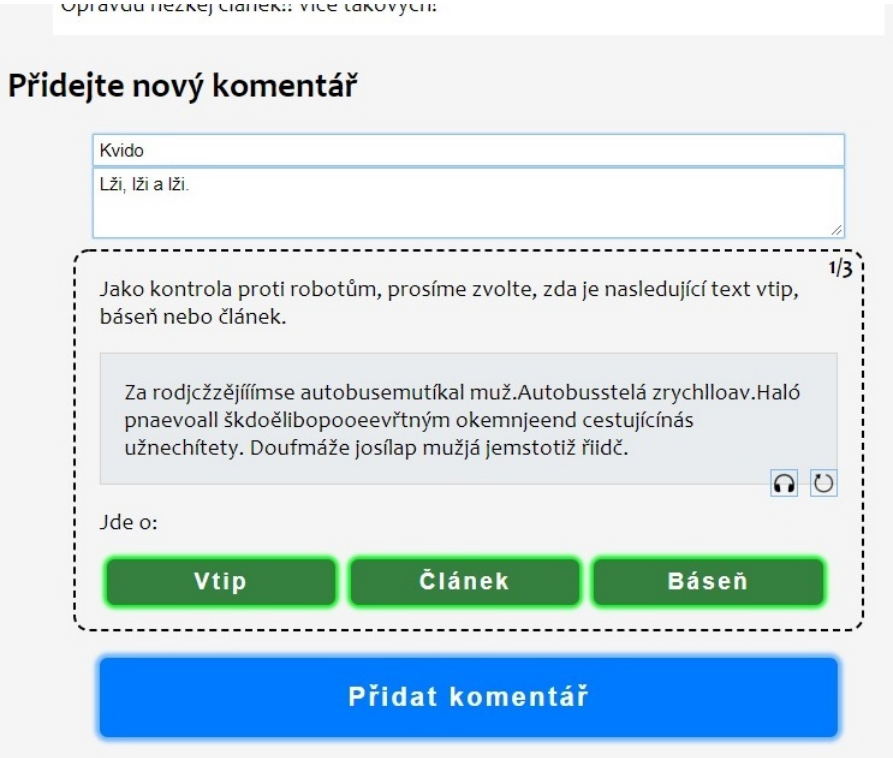
4 APJCaptcha

Značná väčšina CAPTCHA testov sa spolieha na vyriešenie akejsi prirodzene logickej úlohy a neschopnosti neinteligentného stroja orientovať sa v danom probléme. Po predložení algoritmov na riešenie konkrétnej formy úlohy je ale počítač v riešení problému úspešný porovnateľne so skutočným človekom. Vďaka dnešným OCR technológiam si stroj s deformovaným textom dokáže v mnoho prípadoch poradiť a daný text zrekonštruovať. Riešenie textovo-logických úloh je po identifikácii úlohy jednoducho implementovateľné. Je to ale skutočne to, v čom sa líši človek od stroja? V práci bol navrhnutý a implementovaný druh CAPTCHA testu na inom princípe – využívajúci ľudské emócie.
„Čo urobí ako prvé bezdomovec, keď si sadne pred počítač? Pozrie sa, čo je v koši.“
„Čo urobí ako prvé bezdomovec, keď si sadne pred počítač? Pozrie sa na plochu.“
V čom sa líšía tieto dve spojenia otázky a odpovede? Prvá dvojica je (dajme tomu) vtipná, kdežto druhá žiadne emócie nevzbudzuje. Ako to má ale rozlíšiť stroj? Na simuláciu schopnosti rozlíšenia, čo je vtipné a čo nie, by si stroj musel byť vedomý takmer každého aspektu života, ktorého sa vtipy môžu týkať, pospájať si množstvo informácií, vyhodnotiť paradoxy, byť si vedomý vtipných situácií. Na to by si navyše musel plne správne interpretovať význam slov takto textovo zadanej vety. Simulácia tejto schopnosti by bol ťažký problém, dá sa avšak riešiť takáto úloha inak, resp. úplne obísť akékoľvek situačné pochopenie popísujúceho textu. Vyhľadanie daného testovacieho textu, naučenie sa spoločných charakteristík vtipov, analýza typických slovíčok, strojové učenie... Či, a do akej miery je táto CAPTCHA odolná voči takýmto spôsobom prelomenia, sa budeme podrobnejšie venovať vo vyhodnotení robustnosti. APJCaptcha teda bude fungovať na princípe, že používateľovi ponúkne text a on bude mať za úlohu rozpoznať, či ide o vtip, článok alebo báseň. Text bude do voliteľnej miery zdeformovaný, aby bolo pre potencionálneho prelomiteľa ťahšie vyriešiť úlohu pomocou internetových vyhľadávačov. Funkcionalita bude tentokrát predvedená na sekcii stránky umožňujúcej pridávanie komentárov. Po používateľoch nepožadujme registráciu, ale vyriešenie testu. Situácia je znázornená na obrázku 22.



Obr. 22: Sekcia s pridávaním komentárov

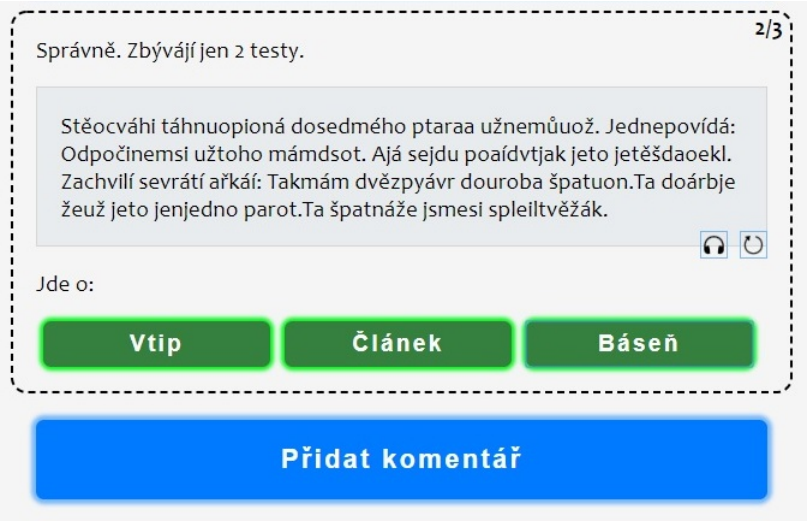
Po pokuse o pridanie komentára pod internetovú diskusiu je používateľ vyzvaný na vyriešenie výzvy. Po overení mu bude umožnené komentár pridať.



Obr. 23: Nová výzva

Ako vidno na obrázku 23, grafické používateľské rozhranie výzvy pozostáva

z hlavného labelu, ktorý vysvetľuje aktuálny stav testu. V pravom hornom rohu sa nachádza informácia upozorňujúca používateľa, koľko správnych odpovedí mu ešte ostáva. Vo farebne odlišnom obdĺžniku sa nachádza samotný deformovaný text hádanky. V jeho pravom dolnom rohu môžme vidieť tlačidlá pre zobrazenie alternatívneho CAPTCHA testu a taktiež tlačidlo na vygenerovanie novej hádanky. Ďalej tlačidlá na voľbu správnej odpovede. Celý tento formulár je ohraničený prerušovanou čiarou, aby bolo jasné, čo patrí ku CAPTCHA testu a čo je súčasť stránky. Programátorovi stránky je samozrejme umožnené pohodlným spôsobom registrovať svoje udalosti, ktoré sa majú stať pri úspešnom vyriešení alebo napríklad pri vyžiadaní alternatívneho CAPTCHA testu.



Obr. 24: Oznámenie o úspešnej odpovedi

Stavový text nás informuje o správnych alebo nesprávnych odpovediach a mení sa každou odpoveďou.

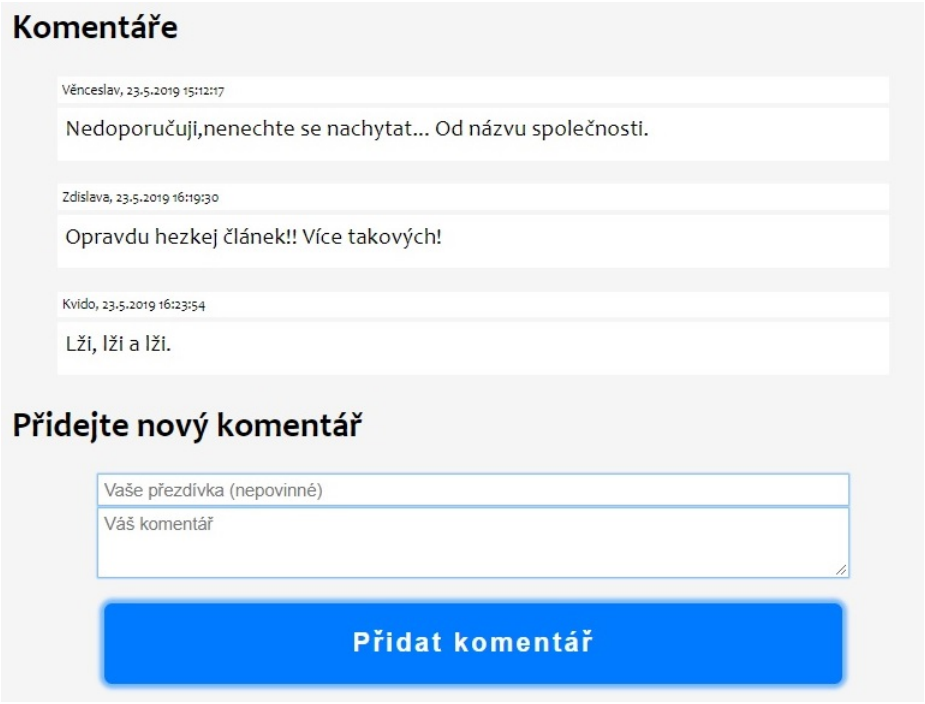
4.1 Použité technológie

4.1.1 Node.js

Na samotné generovanie výziev, ako aj na jednoduchý autentifikátor bolo použité už spomínané Node.js. Tentokrát však nie ako hotová servrová aplikácia, ale ako dva moduly, ktoré je možné do servrových aplikácií pridať. Servrová aplikácia na generovanie dynamickej stránky bola teda pochopiteľne implementovaná taktiež v prostredí Node.js na demonštráciu použiteľnosti týchto modulov.

4.1.2 Express.js

Express.js je webový aplikačný framework pre Node.js. Je navrhnutý na rýchlejší a pohodlnejší vývoj aplikácií pre Node.js. Poskytuje množstvo užitočnej funkcionality ako napríklad routing už v základnom balíčku, či generovanie HTML



Obr. 25: Sekcia s komentármi po vyriešení testu

obsahu pomocou šablónovacieho systému Pug. Express.js bol použitý pri ukážkovej sekcii stránky s hlasovaním.

4.1.3 Java

V práci bol navrhnutý a implementovaný Java 1.7 archív umožňujúci generovanie výziev, spustenie servru poskytujúceho výzvy a jednoduchý autentifikátor. Programovacie prostredie Java tvorí samotný programovací jazyk, ako aj behové prostredie Java Virtual Machine[11]. Java bytekód dokáže na systéme bežať, len ak disponuje JVM, ktoré ho dokáže interpretovať. JVM počas behu monitoruje program a na základe zozbieraných informácii ho dokáže optimalizovať často lepšie, než v čase kompilácie. Preto aj napriek strojovo nekompilovanému jazyku dokážu niekedy Java programy výkonnostne prekonať kompilované jazyky ako C/C++. Java bola od začiatku navrhnutá ako „write once, run anywhere“, sľubuje teda multiplatformovosť.

4.2 Implementácia - Node.js

4.2.1 Generovanie testov

Na generovanie testov slúži balíček `apjcaptcha`. Jeho obsah je nasledovný:

`package.json`
: Informácie o balíčku v JSON formáte.

index.js
: Exportuje triedu APJ. Trieda sa správa obdobne ako manažér predošlého LayerCaptcha. Keďže je exportovaná trieda a nie samotný objekt, manažérov môže byť viac v jednom chode aplikácie.

lib/article-dwer.js
: Exportuje funkciu, ktorá po zavolaní stiahne a poskytne prvý odstavec náhodného článku z českej wikipédie.

lib/poem-dwer.js
: Exportuje funkciu, ktorá po zavolaní stiahne a poskytne báseň zo stránky www.basnicky.cz.

lib/joke-dwer.js
: Exportuje funkciu, ktorá po zavolaní stiahne a poskytne vtip zo stránky www.nej-vtipy.cz.

lib/deformer.js
: Exportuje triedu slúžiacu na deformáciu textu.

lib/txt-tools.js
: Exportuje pomocné funkcie na získavanie textu z HTML dokumentu.

Skripty sťahujúce informácie z webových adries používajú modul `request-promise-native`[16]. Nenájdený, príliš krátky, či inak chybový text v týchto skriptoch vyhodí výnimku, na ktorú reaguje manažér opätovným pokusom o stiahnutie. Hlavné verejne prístupné metódy triedy APJ sú `begin`, `stop` a `popCaptcha` (všetky bez parametrov). Metóda `begin` započne beh programu, t.j. automatické sťahovanie a skladovanie testov, navyšovanie a znižovanie kapacity, `stop` ukončí časovo automatizovaný beh ako napríklad periodická kontrola kapacity. Funkcia `popCaptcha` vráti Promise, ktorý bude splnený ihneď alebo v budúcnosti; podľa toho, či má manažér aspoň jeden test predpripravený.

4.3 Implementácia - Java

V práci bol taktiež navrhnutý a implementovaný Java archív pozostávajúci z tried umožňujúcich generovanie testov, autentifikáciu a beh jednoduchého HTTP servru. Archív je tvorený z dvoch balíčkov: `apj` pre prvé dve zo spomínaných funkcionalít a `apjserver` pre HTTP server.

4.3.1 Balíček apj

Vysvetlime si teraz obsah balíčku `apj` a význam daných verejných tried alebo rozhraní.

[16] https://www.npmjs.com/package/request-promise-native

APJAuthenticator
: Autentifikátor. Funkcionalita, ako aj nastaviteľné parametre sú vysvetlené v sekcii 7.

APJCaptchaProvider
: Slúži na generovanie APJ CAPTCHA testov. Verejné metódy tejto triedy poskytujú funkcionalitu zahájiť a ukončiť generovanie, predpripravovanie testov a samotné poskytovanie s parametrami swap a glue alebo voliteľne s vlastnou funkciou deformujúcou text.

APJFactory
: Vytvorenie novej inštancie predošlých dvoch tried mimo balíčku nie je povolené – konštruktor má modifikátor viditeľnosti „package“. APJFactory dovoľuje vytvárať a následne vracať nové inštancie týchto tried so zadanými parametrami.

AuthPreferences
: Objekt uchovávajúci nastaviteľné parametre pre autentifikátor. Parametre sú totožné s predošlou implementáciou.

AuthRecord
: Finálna trieda obsahujúca informácie o overení subjektu; záznam. Metódy nastavujúce inštančné premenné nie sú mimo balíčka viditeľné.

Captcha
: Trieda predstavujúca trojicu otázka, odpoveď a čas uplatnenia (nie kedy bola otázka vygenerovaná, ale odkedy je pre subjekt aktuálna).

StateFlag
: Enumerátor s možnými hodnotami NEW, LIMIT, TIMEOUT, SUCCESS, MORE, WRONG, ERROR. Predstavuje stav záznamu po pokuse o autentifikáciu.

CaptchaAuthable
: Rozhranie s verejnou metódou `getUniqueId` vracajúca long. Objekt predstavujúci testovaný subjekt musí toto rozhranie implementovať.

TextDeformer
: Rozhranie s verejnou metódou `deform` vracajúcou reťazec akceptujúcou argumenty reťazec `in`, (int) celé čísla `swap` a `glue`. Vlastný použitý „deformer“ musí implementovať toto rozhranie.

CaptchaDeformer
: Predvolená implementácia predošlého rozhrania.

CaptchaObtainer
: Rozhranie obsahujúce verejnú metódu `create` vracajúcu Captcha.

CaptchaJokeObtainer
Rozhranie rozširujúce CaptchaObtainer. Každý vlastný objekt schopný poskytovať vtipy musí toto rozhranie implementovať.

CaptchaPoemObtainer
Rozhranie rozširujúce CaptchaObtainer. Každý vlastný objekt schopný poskytovať básne musí toto rozhranie implementovať.

CaptchaArticleObtainer
Rozhranie rozširujúce CaptchaObtainer. Každý vlastný objekt schopný poskytovať články musí toto rozhranie implementovať.

CaptchaJokeDownloader
Predvolená implementácia CaptchaJokeObtainer. Stará sa o sťahovanie vtipov.

CaptchaArticleDownloader
Predvolená implementácia CaptchaArticleObtainer. Stará sa o sťahovanie článkov.

CaptchaPoemDownloader
Predvolená implementácia CaptchaPoemObtainer. Stará sa o sťahovanie básní.

Ukážme si teraz príklad autentifikácie s použitím implementovaných tried na kóde 5. Najprv APJFactory poskytne inštanciu APJAuthenticator so zvolenými parametrami. Následne zahájime predpripravovanie testov metódou begin. Ďalej sa pokúsime autentifikovať subject s odpoveďou „a“ – article. V premennej subject je uložený ľubovoľný objekt implementujúci CaptchaAuthable. Následne si z vráteného záznamu zistíme stav po pokusu autentifikácie. V if vetve pokračujeme, jedine ak odpoveď bola správna, ale požadujeme ďalšiu správnu odpoveď, teda stav je „MORE“. V posledných dvoch riadkoch len kontrolujeme, či je iný subjekt autentifikovaný.

4.3.2 Balíček apjserver

Vysvetlime si teraz obsah balíčku `apjserver` a význam daných tried. `ReqHandler` je neverejná trieda implementujúca rozhranie Runnable. Inštancie objektov sú spustené v samotatným vláknach znovupoužívaných a manažovaných exekútorom typu cached thread pool. Objekty triedy `Server` už obsahujú verejné metódy na poskytovanie nových CAPTCHA testov. Sú to:

Server(Logger logger, APJCaptchaProvider provider)
Konštruktor triedy s nastaviteľným loggerom a poskytovateľom APJ testov.

Server(Logger logger)
Konštruktor triedy, tentokrát bez nastavenia vlastného poskytovateľa testov - vytvorí sa nový s predvolenými hodnotami.

```
1  APJAuthenticator auth = new APJFactory().setInitCapacity(3)
2    .newAuthenticator(new AuthPreferences().setAnswersRequired(2));
3  auth.begin();
4  // ...
5  AuthRecord rec = auth.tryAnswer(subject, 'a');
6  if (rec.getState() == StateFlag.MORE) {
7    logger.log(Level.INFO,
8      String.format("\%s úspešne zodpovedal otázku, d'alšia je \%s.",
9        subject, rec.getChallenge()));
10 }
11 // ...
12 if(auth.getStateOf(anotherSubject).isAuthd())
13   logger.log(Level.INFO, String.format("\%s je autentifikovaný!",
       anotherSubject));
```

Zdrojový kód 5: Použitie APJAuthenticator spolu s APJCaptchaProvider

`begin(InetAddress ad, int port, int soTimeout)`
: Metóda započne beh servru. Na definovanom porte a adrese otvorí servrový socket. Premenná `soTimeout` udáva hodnotu v milisekundách a udáva ako najdlhšie má počúvajúci servrový socket blokovať.

`terminate()`
: Ukončí beh poskytovateľa testov a zabráni znovuotvoreniu ďalšieho servrového soketu v prípade uplynutia `soTimeout`.

Server odpovedá na požiadavky na danej adrese/porte. Za tým musí nasledovať „/get/formát“, kde formát môže byť xml alebo json. Ďalej môžu nasledovať parametre swap a glue. Pri ich vynechaní vygenerujeme CAPTCHA test, akoby boli tieto parametre nastavené na 1. Je taktiež možné nastaviť kódovanie tela HTTP odpovede pomocou parametru „utf8“ (predvolene je to UTF-16). Pre adresu napríklad http://127.0.0.1:8080/get/xml?glue=2&swap=2&utf-16 teda dostávame odpoveď znázornenú v kóde 6.

```
1  <?xml version="1.0" encoding="UTF-16"?>
2  <root>
3    <question>Poaítčáovč fríenznovaděl (neboilpočítačová feroiznka)je
       vifnomratice oděvtvídigitální forenznívěyd tckjaíýíse háldení
       pvácnírh důkazův pocítačíhča nadahývoct méíihcd.Cílme ješeý
       rtnm zesůobpmforeěznn přečístdaotvé méduimz ddvoůuikentifiadce
       .</question>
4    <answer>a</answer>
5  </root>
```

Zdrojový kód 6: Ukážka odpovede APJ servru

4.4 Deformácia textu

Text sa deformuje za účelom znemožnenia automatizovaného vyhľadania otázky pomocou dostupných vyhľadávačov. Využíva sa idea, že človek je schopný správne interpretovať a porozumieť slovu s prehodenými písmenami. Najjednoduchšie je to pre človeka vtedy, ak prvé a posledné písmeno ostanú na mieste a prehodia sa jedine neokrajové písmená. Algoritmus (predvolenej) deformácie je nasledovný a je rovnaký pre obidve implementácie:

1. získame z textu všetky slová, ktoré sú dostatočne dlhé (ich dĺžka je menšia než MIN_SWAP_LEN)

2. ak je text príliš krátky (počet slov je menší, než TXT_TOO_SHORT), zamieňame písmená vo všetkých slovách. Ak je výmena písmen vypnutá, teda ak je parameter $swap = 0$, tak výmenu preskočíme. Inak budeme zamieňat v n slovách, kde n je náhodné celé číslo medzi štvrtinou počtu slov a počtom slov. Ak je $swap > 1$, miesto štvrtiny pri výpočte náhodného čísla použijeme polovicu.

3. vyberieme n náhodných slov a v každom slove zameníme 2 neokrajové písmená. Ak je slovo dlhé (je o 3 písmena dlhšie, než stanovené minimum na výmenu), znovu zameníme 2 neokrajové písmená.

4. ak je zapnuté lepenie slov, zvolíme si každé koľké slovo nalepíme na predošlé (nedáme medzi nich medzeru). Ak je parameter $glue > 1$, lepíme každé druhé slovo, inak každé tretie. Ak je počet slov menší, než dvojnásobok TXT_TOO_SHORT, lepíme každé.

4.5 Robustnosť APJCaptcha

Výber jednej z troch možností činí pre útočníka slušnú šancu na uhádnutie. Otázka by preto nikdy nemala byť len jedna. n otázok znamená šancu na uhádnutie $\frac{100}{3^n}\%$. Teda napríklad pri 3 otázkach je to šanca 3,7%, čo už môže predstavovať dostatočne vysokú robustnosť, no aj zvýšené nepohodlie ľudí, ktorí sa test pokúšajú vyriešiť.

Najlepší spôsob, ako tento CAPTCHA test prelomiť je skúsiť hľadať v texte kľúčové slová charakteristické pre jednotlivé kategórie. Nájdime napríklad v texte podreťazce „chuck“, „chcuk“, „cuhck“ alebo „ccuhk“ a môžme vsadiť na to, že pôjde o vtip s Chuckom Norrisom. Podobne pre „pepíček“, „zajíček“, „cikán“, ale aj „manželka“, či „povídá“. Pre článok môžu byť charakteristické dátumy. Nájsť číslo napríklad 1981 alebo 1891 znamená vysokú pravdepodobnosť, že pôjde o článok. Napriek tomu, že pri básniach by sa taktiež dali nájsť charakteristické kľúčové slová, z týchto troch kategórii je rozpoznať báseň asi najťahšie. Preto by útočník mohol voliť túto odpoveď vždy, ak nenájde kľúčové slová pre ostatné kategórie. I nájdenie kľúčového slova pre danú kategóriu v deformovanom textu však správnu odpoveď nezaručuje. Naviac, je len obmedzené množstvo pokusov.

Z pohľadu používania správnych odpovedí je táto CAPTCHA dostatočne robustná. Počet básní, vtipov a článkov sa pre každú kategóriu pohybuje v tisíckach. Text je náhodne deformovaný a dokonca, pri príliš dlhých textoch, skracovaný o náhodné číslo. Len ukladanie si známych odpovedí by teda nemalo zmysel.

5 Illucaptcha

Ďalšia CAPTCHA je vylepšením dobre známej kombinácie obrázkovej a textovej úlohy: obrázková úloha spočíva v rozpoznaní tvarov a farieb objektov na danom obrázku, kdežto textová predstavuje jednoduchú matematickú úlohu a to spočítať dva druhy tvarov o zadanej farby. Tvary sú nemenné obrázky vpisované do obrázka náhodne. Všetky tvary sú rovnakej farby. „Farba“ je vytvorená pomocou optického klamu, ktorý funguje nasledovným spôsobom: pozadie tvoria riadky trojice RGB farieb. Po riadkoch prekresluj pozadie tvarmi, ale preskakuj riadky, ktoré majú farbu pozadia rovnakú, ako má zdanlivo mať daný tvar. Takto pruhované tvary sa už používateľovi javia ako farebné. Obrázok je dostatočne malý (predvolene 300x300), aby ho webový prehliadač nemusel zmenšovať a zmenšovacím algoritmom nepokazil ilúziu. Funkcionalitu znázornime na ukážke fiktívnej časti stránky, kde je pre presmerovanie na adresu s uloženým súborom na stiahnutie podmienené CAPTCHA overením.

Podobne ako u predošlých testoch, používateľ je informovaný o správnosti jeho odpovede, koľko otázok mu ostáva, atď. Po vyriešení výzvy je presmerovaný na adresu „/download“, kde by už na ňho mohol čakať pripravený súbor na stiahnutie. Pre úplnosť ukážky je neoverený návštevník tejto adresy presmerovaný na úvodnú stránku s automaticky otvoreným CAPTCHA testom.

5.1 Použité technológie

5.1.1 Python

V práci bol vytvorený modul „ilcaptcha“ pre Python 3. Python je vysokoúrovňový skriptovací programovací jazyk, navrhnutý v roku 1991. Disponuje dynamickej kontrole datových typov, je multiparadigmový[12]. Podporuje objektovo orientovaný štýl programovania, imperatívny, procedurálny alebo funkcionálny štýl. Výhodou je široký arzenál dostupných balíčkov. V práci bol (okrem vstavaných modulov) použitý modul „pypng“[17] na de/kódovanie obrázkov PNG formátu. Ďalším dôvodom použitia práve Pythonu bola osobná motivácia autora práce – vyskúšať niečo napísať v Pythone.

[17] https://pypi.org/project/pypng/

Obr. 26: Illucaptcha GUI

5.1.2 Flask

Triviálna webová stránka je poskytovaná pomocou webového frameworku Flask. Flask bol zvolený pre jeho minimalistickú povahu v porovnaní s inými frameworkmi [13] ako napríklad Django. Na naše účely je teda viac, než dostačujúci.

5.2 Implementácia

Samotný modul `ilcaptcha` poskytuje funkcionalitu manažéra obdobnú predošlým implementáciám. Generovanie funguje obdobne ako v predošlých implementáciach – negeneruje sa pri požiadavke, ale uchovávame si predpripravené testy. Ukážme si teraz „verejné“ metódy modulu.

`begin()`

Započne generovanie a periodickú kontrolu kapacity testov.

terminate()
Zastaví generovanie a periodickú kontrolu kapacity testov.

take()
Vráti CAPTCHA test vo forme objektu slovníka, kde pre kľúč „question“ je hodnota zoznam dvojíc farba-tvar, ďalej v „answer“ je číselná odpoveď a v path sa nachádza relatívna cesta k PNG súboru. Nakoniec „deploy_time“ je timestamp vrátenia objektu. Funkcia blokuje, kým aspoň jeden test nie je k dispozícii. Ak je čakateľov viac, odpovedá sa spôsobom FCFS.

Modul obsahuje nastaviteľné premenné CAPACITY, CHECK_INTERVAL, DYNAMIC_CAPACITY, CUTBACK_PERIOD, MAX_CAPTCHA_AGE, CLEANUP_ON_EXIT, ktorých význam je rovnaký ako pri predošlých implementáciach. Súčasťou modulu je taktiež submodul – obdoba autentifikátora – authr.py s nastaviteľnými premennými autentifikácie a generator.py. Nastaviteľné premenné generátora sú nasledovné:

SHAPERES
Rozmer zdrojových obrázkov všetkých tvarov. Predpokladá sa, že obrázky sú výlučne štvorcového tvaru. Predvolene 40.

MINSHAPENUM
Minimálny počet tvarov vpísaných do obrázka. Predvolene 11.

MAXSHAPENUM
Maximálny počet tvarov vpísaných do obrázka. Predvolene 16.

PICW
Šírka obrázka v pixeloch. Predvolene 300.

PICH
Výška obrázka v pixeloch. Predvolene 300.

Pri zmene počtu tvarov sa musí programátor uistiť, že sa pri náhodnom rozložení do obrázka tvary vždy zmestia. Zoznam tvarov a cesta k ich obrázkom sú uložené v shapes.json.

5.3 Robustnosť Illucaptcha

Zo 100 vygenerovaných testov bola takáto frekvencia správnych odpovedí:

Odpoveď	Početnosť
2	9
3	18
4	17
5	17
6	18
7	12
8	4
9	3
10	0
11	2

Dajme tomu, že útočník vždy háda najpočetnejšiu správnu odpoveď – 3. To predstavuje šancu uhádnutia približne $100(\frac{18}{100})^n\%$, kde n je počet požadovaných správnych odpovedí. Pre $n = 2$ je to teda $3,24\%$ alebo $0,58\%$ pre 3 požadované správne odpovede. Ak teda požadujeme aspoň dve správne odpovede, považujeme overovanie za robustné voči hádaniu správnych odpovedí.

Používanie známych správnych odpovedí nepredstavuje pre Illucaptcha hrozbu. Do obrázka sú tvary vpisované o náhodnom počte, náhodných farbách, do náhodných pozícií. Z tohto pohľadu je Illucaptcha dostatočne robustná.

Illucaptcha je pomerne zraniteľná voči prelomeniu strojovým učením. Identifikácia tvaru z obrázka je jedna z najjednoduchších úloh OCR. Taktiež vytvorená pseudofarba je len skupina farebných čiar prechádzajúcich cez daný tvar. Z tohto pohľadu tak CAPTCHA nedisponuje dostatočnej robustnosti. Z tohto dôvodu je dobrý nápad používať Illucaptcha len v situáciach, kde sa pokusy o prelomenie overovania takýmto spôsobom neočakávajú.

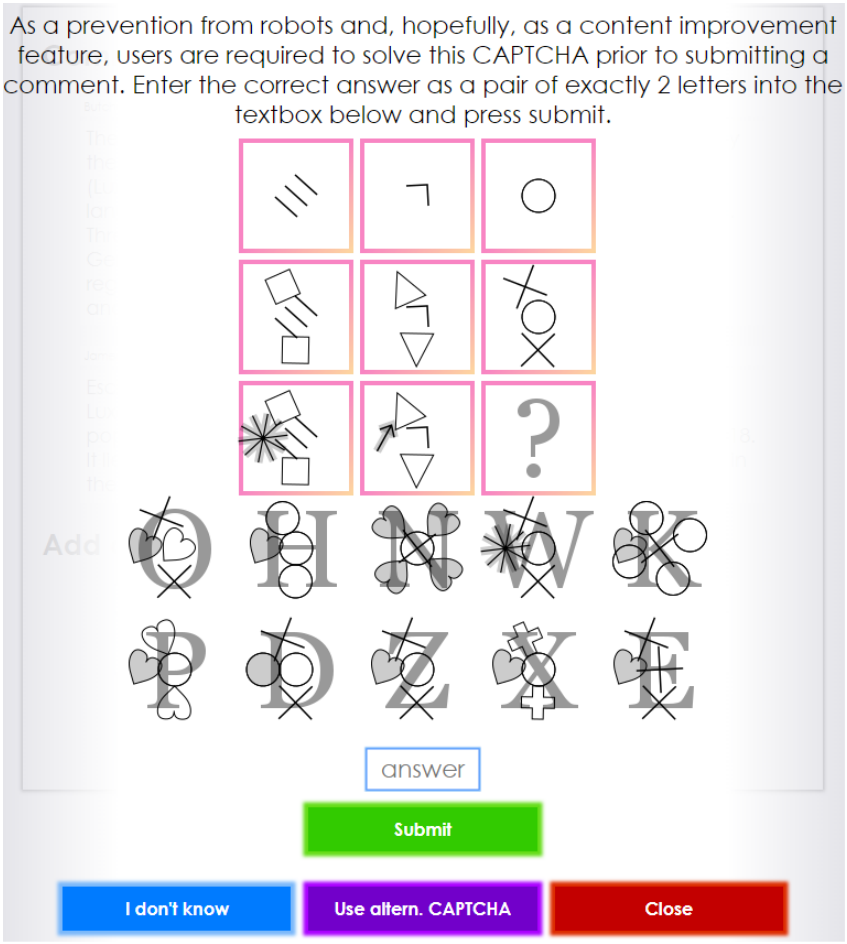
6 IQCaptcha

Takmer každá CAPTCHA a všetky naše doterajšie navrhnuté a implementované CAPTCHA testy sa snažia overiť používateľa takým spôsobom, aby bolo overenie dôveryhodné, ale rýchle a pre používateľa jednoduché. Existujú však prípady, kde je opodstatnené vynechať posledný prepoklad a overiť len používateľov ochotných riešit komplexnejšiu úlohu, vyfiltrovať teda nie len automatizované akcie, no taktiež „lenivých“ používateľov.

Jedným z takýchto prípadov overenia by mohlo byť pridávanie komentárov pod články alebo videá. Opodstatnenie je tu dobre známe – hľadať kvalitný príspevok v sekcii komentárov pod akýmkoľvek populárnym internetovým obsahom je ako hľadať ihlu v kope sena. S ubúdajúcou potrebnou dobou a s čoraz pohodlnejšími možnosťami komentovania pribúdajú nekvalitné príspevky. Hodnotenie komentárov a zobrazovania len najlepšie hodnotených nefunguje – nezriedka vyplávajú hore práve tie menej kvalitné. Takmer ako keby sa hodilo overiť disku-

tujúcich krátkym IQ testom.

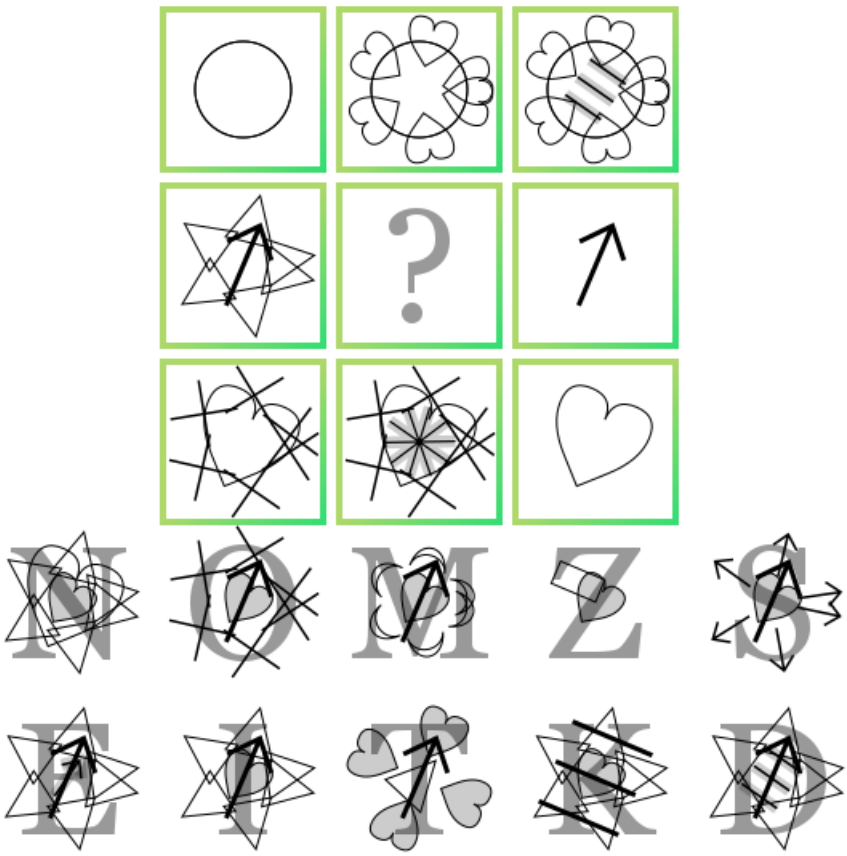
IQCaptcha slúži práve na takéto overovanie. Predstavuje spôsob, ako napríklad filtrovať ľahkomyselne priepievajúcich diskutujúcich. Na demonštráciu recyklujem stránku so sekciou umožňujúcou pridávanie komentárov zobrazenú na obrázku 22. Ak nie je používateľ overený, tak po odoslaní komentáru sa používateľovi otvorí modálne okno s testom ako na obrázku 27.



Obr. 27: IQCaptcha GUI

IQCaptcha spočíva v logickej úlohe doplniť možný chýbajúci obrázok. Princíp je nasledovný. Majme 3 skupiny obrázkov, každá skupina pozostáva z 3 obrázkov. Každá skupina začína s práve 1 typom tvaru, no nie výlučne jedným kusom tvaru. V každom ďalšom obrázku skupiny pribudne práve jeden nový typ obrázku (ktorý sa ešte v skupine nenachádza) dokreslený s určitými parametrami. Tieto parametre však zdieľajú pri kreslení nových typov všetky skupiny obrázkov. Skupiny obrázkov su umiestnené do 3x3 mriežky buď po riadkoch alebo po stĺpcoch. Následne je tretí obrázok v jednej zo skupín nahradený otáznikom a používateľ má hádať, čo by sme miesto otáznika mohli doplniť, aby to súhlasilo s parametrami umiestňovania a pridávania obrázkov. Pri danej implementácii sú

z 10 vygenerovných možností vždy 2 správne a používateľ musí zadať obe. Hádanka je vždy v poslednom obrázku zo skupiny. Každá vygenerovaná možnosť má cez seba prekreslené písmeno, ktoré musí naviac používateľ identifikovať a prepísať, ak súhlasí so správnou odpoveďou. Ukážme si teda riešenie na nasledovnom príklade:



Obr. 28: IQCaptcha

3 skupiny sú umiestnené v riadkoch. To vieme, pretože v prvom obrázku sa nachádza jediný tvar – kružnica a to teda znamená, že existujú ešte minimálne 2 obrázky v celom teste a ešte práve 2 obrázky v jej skupine, ktoré obsahujú takto veľkú kružnicu o takýchto súradniciach. Obrázok pod kružnicou kružnicu neobsahuje, teda skupiny sú v riadkoch. V ďalšom kroku je pridaných 6 nevyplnených sŕdc po obvode kružnice. Vidíme, že 2 srdcia sa v prvom a štvrtom kvadrante prekrývajú. Následne sú vpísané 3 „vyplnené“ čiary do stredu obrázka. To sa teda bude diať pre každú skupinu, len s inými tvarmi a obrázky nemusia nutne byť v rovnakom poradí ako táto skupina. Príkladom je posledný riadok, prvý krok sa nachádza v mriežke vpravo, druhý vľavo a tretí v strede. Ajkeď je poradie rozhodené, tvary sa pridávajú rovnakým spôsobom: najprv jednoduché nakreslenie (srdca), potom 6 „plusov“ (kde sa 2 znova prekrývajú) a následné vpísanie vyplneného tvaru pripominajúceho slnko. Prechádzajme cez ponúkané možnosti:

N: vieme, že prvý pridaný tvar bola šípka ukazujúca približne 80 stupňov a následne 6 trojuholníkov. Táto odpoveď teda nie je správna: v obrázku chýba šípka

O: šípka je na správnom mieste, dokonca aj nový vložený tvar by mohol byť správny. Avšak trojuholníky sú v tejto možnosti nahradené tvarmi tvaru +

M: prípad podobný predošlému, trojuholníky boli nahradené tvarmi mesiaca

Z: tento prípad je úplne mimo: chýba tretí tvar a ani jeden z týchto dvoch tvarov neodpovedá daným dvom v hádanke

S: chýbajúce trojuholníky

E: trojuholníky aj šípka sú na svojom mieste, avšak novodoplnený tvar je opäť šípka, čo je v rozpore s tým, že sa do obrázku pridávajú len tvary, ktoré sa tam nenachádzajú. To musí hádajúci vydedukovať z dvoch vyplnených skupín. Ak si hádajúci nie je istý, či sa tvary vo vyplnených skupinách neopakovali zámerne alebo kvôli náhode, pomôže mu k tomu vedomosť, že počet správnych odpovedí je práve dva.

I: šípka aj trojuholník odpovedá. Pridaný tvar sa v obrázku ešte nenachádza a je vyplnený. Budeme to teda považovať za správnu odpoveď.

T: obrázok obsahuje šípku aj trojuholník, avšak nie o správnom počte (a ani nerotuje).

K: obrázok neobsahuje šípku

D: obrázok obsahuje šípku a trojuholníky o správnom počte, s novým vyplneným tvarom v strede

Používateľ teda zadá do textového poľa napríklad „iD“, odošle odpoveď a zobrazí sa mu diskusia s jeho pridaným komentárom. Zadané písmená môžu byť v ľubovoľnom poradí a nezáleží na tom, či sú veľké alebo malé.

6.1 Použité technológie

6.1.1 Canvas

Canvas technológia s príchodom HTML5 umožnila programátorom vytvárať interaktívne HTML prvky <canvas> schopné kresliť 2D grafiku. Do prvku je možné pomocou JavaScriptových príkazov vkresľovať obdĺžniky, text, komplexnejšie tvary pomocou ciest, obrázky, výplne, atď. Canvas môže na webe poslúžiť pre úpravu obrázkov, vizualizáciu dát, hry alebo videá[14]. Canvas predstavuje pohodlný a kvalitný spôsob generovania 2D grafiky, preto tu bolo jasnou voľbou. V práci boli obrázky generované na strane servra pomocou Node.js modulu

node-canvas[18]. Obrázky su generované na strane servra, používateľov prehliadač teda nepotrebuje podporovať HTML5.

6.2 Implementácia

V práci bol implementovaný Node.js modul iqcaptcha, ktorý sa chová obdobne ako manažér z predošlých implementácií, teda predpripravuje si testy, dynamicky mení kapacitu, poskytuje predpripravené testy. Ďalej sa využíva spomínaný modul captcha-authr (sekcia 7).

Balíček iqcaptcha obsahuje súbory index.js, geometry.js, provider.js a package.json. Skript súboru index.js exportuje triedu IQC umožňujúca vytvoriť inštanciu manažéra, geometry.js obsahuje zoznam tvarov, ako aj metódy logiky kreslenia, či generovania tvarov a ich parametrov. Skript provider.js s jedinou funkciou `provide()` už využívajúc geometry.js vráti hotový CAPTCHA test vo forme objektu s kľúčmi `choices` uchovávajúc pole písmen zobrazených v možnostiach, `answer` s hodnotou reťazca práve dvoch písmen z `choices` a `data` s reťazcom odpovedajúcim Base64 kódovaním Canvas obrázka. Ďalej `provider.js` má konštanty udávajúce proporcie vygenerovaného obrázka `tileRes`, `elementRes`, `padding`, `canvasWidth` a `canvasHeight`. Prvá zo spomínaných konštánt určuje veľkosť jednej oblasti z hlavnej 3x3 mriežky a konštanta `elemRes` určuje veľkosť obrázka, konkrétne počiatočnú veľkosť vpísaných tvarov, zmenšujúcu sa po vrstvách. Všetky tieto konštanty sú predvolene závislé od `tileRes`, jej zmena vhodne ovplyvní ostatné rozmery. Skript `geometry.js` má nasledovné konštanty:

`FILLSTYLE`
: Štýl výplne tvarov. Predvolene reťazec `'rgba(0, 0, 0, 0.2)'`.

`LETTER_FILLSTYLE`
: Štýl výplne písmen. Predvolene reťazec `'rgba(0, 0, 0, 0.4)'`.

`MIN_CLOCK_ITERS`
: „clock“ je názov funkcie ktorá v jednom kroku po kružnici umiestni náhodný počet tvarov toho istého typu. Konštanta teda udáva minimálny počet umiestnených tvarov[19].

`MAX_CLOCK_ITERS`
: Udáva maximány počet „hodinovo“ umiestnených tvarov.

`POSSIBLE_LETTERS`
: Pole možných písmen, ktoré budú použité ako možnosti a následne nimi možnosti prekreslené (teda spodná 2x5 mriežka). Predvolene je to pole po `'ADEIHKMNOPSTWXZ'.split('')`, teda nie každé písmeno abecedy. Dôvod

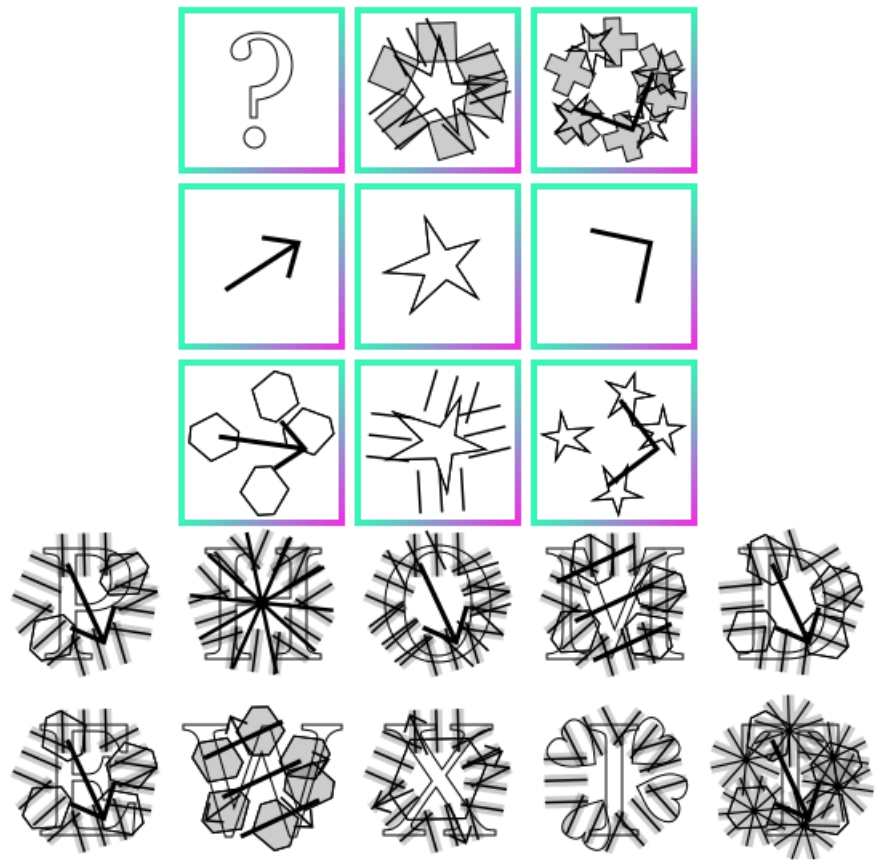
[18] https://www.npmjs.com/package/canvas

[19] v skutočnosti ešte existuje šanca, že vykreslovanie niektorého z tvarov preskočíme, teda je možné, že uvidíme o 1 menší počet vykreslených tvarov, než udáva táto konštanta

neobsiahnutia každého písmena je podobnosť niektorých písmen. Napríklad C sa podobá na G, P na R, čo by mohlo riešiteľa uviesť do omylu.

TEXT_FILL
Určuje, či majú byť písmená možností vyplnené. Predvolene false.

ROTATE_PER_LAYER
Určuje, či majú byť každé už vložené tvary pri novom vkladaní rotované o náhodný uhol. Predvolene false. Ukázka takto vygenerovaného testu je na obrázku 29.



Obr. 29: IQCaptcha s postupným rotovaním a bez vypĺňania textu

6.3 Robustnosť IQCaptcha

Keďže správnych odpovedí je vždy práve 2, šanca na uhádnutie je $\frac{2}{15} \cdot \frac{1}{14} \doteq 0,95\%$. Používame totiž vždy 10 náhodných písmen z (predvolene) 15 možných. Kritérium robusnosti z pohľadu náhodného hádania teda CAPTCHA spĺňa dostatočne.

Znovupoužívanie správnych odpovedí je pri IQCaptcha takmer nemožné. Možných generovaných tvarov je síce len 14, no sú umiestňované v každom obrázku

náhodne, pod náhodnym uhlom, náhodným spôsobom a to isté platí aj pre obrázky možností. Navyše písmená sú vyberané a umiestňované náhodne a taktiež skupiny môžu byť umiestnené horizontálne alebo vertikálne s prehodeným poradím či už samotných skupín, alebo samotných obrázkov skupín. Všetky tieto aspekty generujú takmer nevyčerpateľné množstvo možností. Tento CAPTCHA test je teda z pohľadu učenia sa známych odpovedí dostatočne robustný.

Prelomenie tohto CAPTCHA testu s využitím OCR techník na detekciu tvarov by bolo výrazne komplikovanejšie, než pri predošlej Illucaptcha. Tvary majú rôzne dĺžky, sú otočené pod rôznym stupňom, prekrývajú sa a navyše sú cez možnosti prekreslené písmená, ktoré treba taktiež identifikovať. Azda najjednoduchším spôsobom by bolo nájdenie v poradí druhého obrázku v skupine, v ktorej sa nachádza hádaný obrázok. Následne by sa porovnával tento obrázok s možnosťami a ako odpoveď by sa zvolili prvé dve možnosti, ktoré obrázok otočený o nejaký uhol obsahujú. Pre obtiažnosť implementácie prelomenia, jeho predpokladanú nízku úspešnosť a jednoduchosť pridania ďalšieho možného tvaru budeme avšak považovať tento CAPTCHA test dostatočne robustný aj z tohto hľadiska.

7 Autentifikátory

V práci boli navrhnuté a implementované skripty odkrývajúce pohodlné API pre autentifikáciu používateľov. Konkrétne 4 rôzne autentifikátory: triedu pre PHP, pre platformu Java, modul pre Python a Node.js. PHP trieda je navrhnutá tak, že každý overovaný používateľ má mať jednu jej inštanciu. Tá pre neho eviduje interakciu s CAPTCHA testom. Ostatné fungujú na princípe centrálneho autentifikátora a každý používateľ má záznam. Autentifikátory majú vo všeobecnosti tieto nastaviteľné parametre:

`DROP_WRONG_AFTER`
: Po akom časovom limite sa používateľovi odpustí jedna nesprávna odpoveď.

`REQUIRED_ANSWERS`
: Počet požadovaných odpovedí, po ktorom je overenie úspešné.

`MAX_WRONG_ANSWERS`
: Počet nesprávnych odpovedí, po ktorom musí používateľ čakať dobu DROP_WRONG_AFTER, než sa môže pokúsiť vyriešiť test.

`RESET_ANSWERS_ON_WRONG`
: Booleovská hodnota určujúca, či pri sa má používateľov počet správnych odpovedí vynulovať pri nesprávnej odpovedi.

`ANSWER_TIMEOUT`
: Udáva maximálnu dobu, ktorú má používateľ na vyriešenie jednej výzvy.

TOO_FAST
: Udáva dobu, ktorá je považovaná za príliš krátku na to, aby sme verili, že ide skutočne o legitímny pokus o zodpovedanie testu.

ON_REGEN_WRONG
: Desatinné číslo udávajúce hodnotu, o ktorú sa má zvýšiť počet používateľovych nesprávnych odpovedí pri požiadaní o nový test. Nemá zmysel, aby toto číslo nebolo z rozsahu 0 až 1. Predvolene všade 0,5.

ON_TOOLONG_WRONG
: Desatinné číslo udávajúce hodnotu, o ktorú sa má zvýšiť počet používateľovych nesprávnych odpovedí pri vypršaní časového limitu na odpoveď. Predvolene 0,5.

AUTH_TIMEOUT
: Časová doba, po ktorej vyprší používateľovo overenie.

Node.js modul autentifikátor je spoločný pre všetky predstavené CAPTCHA testy. Balíček captcha-authr má nasledovnú štruktúru:

package.json
: Informácie o balíčku v JSON formáte.

index.js
: Exportuje triedu CaptchaAuthr.

AuthRecord.js
: Exportuje triedu AuthRecord. Objekty tejto triedy sú vlastne záznamy autentifikácie a spravuje ich CaptchaAuthr. Priama manipulácia s nimi nie je mimo CaptchaAuthr potrebná.

Konštruktor CaptchaAuthr požaduje argumenty `provider` a objekt s parametrami `authPreferences`. Argument `provider` je poskytovateľ CAPTCHA testov. Môže to byť akýkoľvek objekt s asynchrónnou (vracajúcou Promise) metódou s názvom „popCaptcha“. Argument `authPreferences` sú spomínané nastaviteľné parametre. Okrem týchto parametrov je možné registrovať vlastnú funkciu kontrolujúcu správnu odpoveď a taktiež funkciu poskytujúcu otázky z CAPTCHA objektu vráteného providerom, keďže generované testy môžu byť rôzne a mať teda odlišné atribúty a postupy kontrolovania otázok. Príklad započatia behu IQCaptcha, jeho jednoduché spojenie s autentifikátorom a definovanie vlastných parametrov je vidno v kóde 7.

Všetky parametre sú voliteľné. Ak nejaký definovaný nie je, použije sa jeho predvolená hodnota. Udržuje sa mapa párov unikátny kľúč a záznam – objekt triedy AuthRecord. Trieda CaptchaAuthr má nasledovné metódy:

tryAuth(id,ans)
: Pokúsi sa autentifikovať subjekt s jednoznačným identifikátorom `id` s odpoveďou `ans`. Ak odpoveď chýba a neexistuje ešte o subjekte záznam, vytvorí

```
const iqc = new IQC();
iqc.begin();
const authPreferences = {
   customChecker: (ans1, ans2) => String(ans1).split('').sort().join
       ('').toLowerCase() === String(ans2).split('').sort().join('')
       .toLowerCase(),
   customChallenger: captcha => (captcha.data),
   requiredAnswers: 1,
   resetOnWrong: true,
   answerTimeout: 60000
};
const authr = new CaptchaAuthr(iqc, authPreferences);
```

Zdrojový kód 7: Ukážka spojenia autentifikátora captcha-authr s IQCaptcha

sa nový a vygeneruje sa test. Vždy sa odpovie vo forme objektu, ktorý obsahuje informácie o stave autentifikácie a otázku CAPTCHA testu. Ak je odpoveď reťazec „regen“, odpovie sa novovygenerovaným testom.

deAuthAndGenNew(id)
: Vygeneruje pre subjekt s daným identifikátorom nový záznam a následne anuluje jeho overenie. Vráti informácie o zázname vo forme spomínanej v predošlej metóde.

deAuth(id)
: Zmaže záznam pre subjekt s daným identifikátorom.

isAuthd(id)
: Vráti booleovskú hodnotu určujúcu, či je je subjekt s daným identifikátorom autentifikovaný.

authSucceeded(stateVal)
: Vráti booleovskú hodnotu určujúcu, či pokus o autentifikáciu bol úspešný (subjekt odpovedal správne) a hotový (na všetky výzvy). Ako argument očakáva objekt so stavovými informáciami vrátený metódami `tryAuth` alebo `deAuthAndGenNew`.

Autentifikácia využívajúca tieto balíčky by teda mohla vyzerať ako na zdrojovom kóde 8. Premenná `req` bude identifikovať subjekt jednoznačne formou identifikátora express-session sedenia – `sessionID`. Premenná `answer` bude zaslaná odpoveď (alebo `undefined` ak ešte odpoveď zaslaná nebola), do `authrResponse` si po čakaní na splnenie Promisu uložíme odpoveď funkcie pokúšajúcej sa o autentifikáciu `tryAuth`. Ak bola autentifikácia úspešná a hotová, pridá sa komentár a do odpovede sa zaznačí, že bol komentár pridaný úspešne. Nakoniec sa odpovie stavom o autentifikácii.

```
const processRequest = async req => {
   const [id, answer] = [req.sessionID, ((req.body || {}).captcha ||
       {}).answer],
      authrResponse = await authr.tryAuth(id, answer);
   if (authr.authSucceeded(authrResponse)) {
      const cmt = req.body.comment; // comment consists of nick and
          body
      if (cmt && cmt.body) {
         comments.push({date: getDate(), nick: cmt.nick || "Anonym",
             body: cmt.body});
         authrResponse.comment = 'success';
      }
   }
   return authrResponse;
};
```

Zdrojový kód 8: Ukážka použitia Node.js autentifikátora

8 Klientské skripty pre spracovávanie a odosielanie požiadaviek

V práci bolo taktiež pre každý druh CAPTCHA testu implementované grafické používateľské rozhranie. Grafické používateľské rozhrania sa skladajú z jedného CSS súboru obsahujúci kaskádové štýly zodpovedné za grafickú stránku rozhrania. CSS selektory vo vnútri sú stanovené tak, aby sa minimalizovala šanca konfliktu s inými štýlmi, ktoré môže správca stránky do dokumentu pridať. Selektor každého elementu, pseudotried a tried je podmienený triedou s názvom odpovedajúcim danému CAPTCHA testu. Napríklad v súbore ilcGui.css `.ilcaptcha .ilcMiniBtn:nth-child(2n+1)` pre každé v poradí nepárne malé tlačidlo v Illucaptcha. V adresári každej ukážky sa taktiež nachádza ešte jeden CSS súbor na všeobecnú štylizáciu webstránky, ktorá slúžila ako príklad.

Grafické rozhrania sú navrhnuté tak, že ich HTML prvky sa vložia do DOM až po tom, čo ich je treba. Správcovi stránky teda stačí vložiť prázdny div element s určenou triedou, ak rozhranie nie je modálne okno. Ak je, potrebné HTML prvky sa vytvoria na konci dokumentu. Funkcionalitu vloženia potrebných prvkov zabezpečuje pre každý CAPTCHA test zahrnutá knižnica – skript, ktorý je potrebné zahrnúť na koniec dokumentu. Po spustení vytvorí objekt `CaptchaResponseParserFactory`. Názov je dosť dlhý a špecifický na to, aby nekolidoval s inou globálnou premennou. Objekt vracia API s očakávanou funkcionalitou. Okrem iného ešte obsahuje možnosť odosielať odpovede na CAPTCHA testy vo formáte, ktorému rozumie backendová implementácia, no hlavne interpretovať jej spätné odpovede a reagovať na ne pomocou udalostí, ktoré si na ňom vieme registrovať. Používateľ skriptu je tak odklonený od potreby skladania a parsovania odpovede do vhodného formátu. Niektoré udalosti, napríklad zmena stavového textu, či automatické otvorenie/zatvorenie modálneho okna sú už predvolene

nastavené. Taktiež sú predvolene nastavené texty zobrazené používateľovi. Až na APJ Captcha sú všetky v angličtine. Tieto malé knižnice sú od začiatku pripravené splňovať svoju funkcionalitu bez zadávania parametrov, či špecifikácie vlastných funkcií. Daň za to je ale niekoľko odlišných knižníc, ktoré su z veľkej časti rovnaké. Globálny menný priestor obsadíme teda len jednou premennou. V našich ukážkach s ňou pracujeme v IIFE[20] rozsahoch. Konkrétne hovoríme o knižniciach apjScript.js, iqcScript.js, ilcScript.js a lcScript.js. Pozrime sa teraz na niektoré spoločné odkryté metódy a atribúty týchto knižníc:

`postAddress`
: Adresa, na ktorú budú smerovať naše AJAX požiadavky. Predvolene '/'.

`postData`
: Objekt, ktorý bude ako JSON odoslaný na server. Veci spojené s CAPTCHA overovaním sa budú nachádzať v jeho atribúte „captcha“. Správca si môže do objektu uložiť informácie, ktoré budú spolu s odpoveďami na test odoslané tiež. Napríklad hodnotu hlasovania.

`imgsPath`
: Cesta k statickým obrázkom. Pri stavbe GUI testu môžme totiž požadovať obrázky. Predvolene 'images/' alebo 'static/' pre ilcScript.js.

`_elems`
: Objekt uchovávajúci vytvorené HTML prvky GUI.

`hdrContentType`
: Hodnota „Content-Type“ hlavičky požiadavky. Predvolene 'application/json'.

`sentPrefix`
: Reťazec za ktorý bude konkatenovaný výsledný JSON reťazec. Predvolene prázdny reťazec.

`defaultLabels`
: Objekt uchovávajúci textové popisy GUI. HTML prvky budú vytvorené s týmito textovými hodnotami.

`show()`
: Zostrojí a ukáže GUI. Ak už zostrojené bolo (HTML prvky sa v DOM nachádzajú), odkryje ich, ak bolo skryté.

`hide()`
: Skryje GUI, ak bolo zostrojené.

`processAnswer()`
: Spracuje odpoveď zo servra a podľa udalosti zavolá odpovedajúce metódy.

[20] Immediately Invoked Function Expression

handlers
: Objekt s atribútmi uchovávajúcimi polia funkcií, ktoré sa zavolajú pri daných udalostiach.

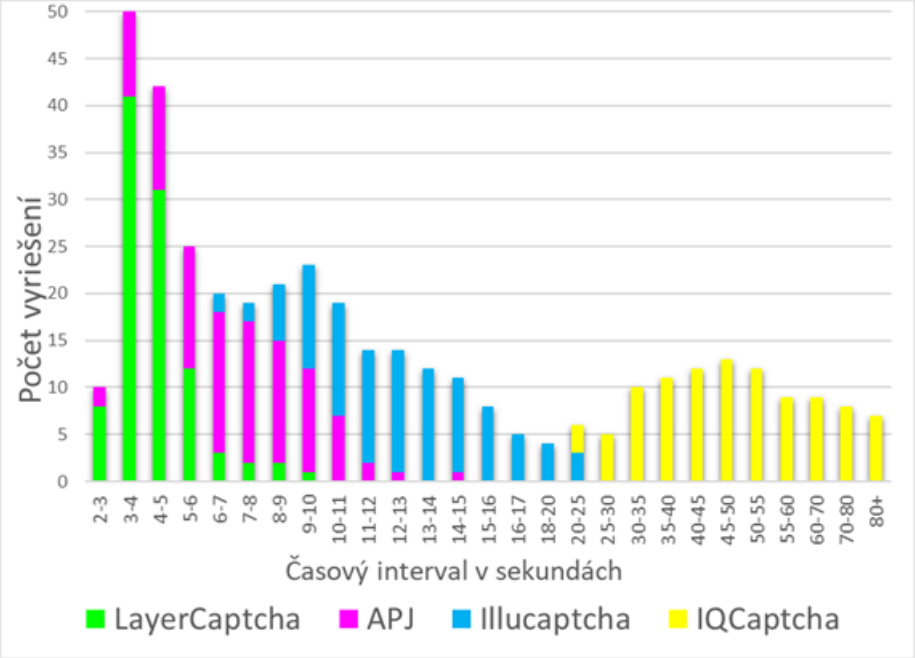
Posledný objekt dovoľuje správcovi registrovať funkcie, ktoré sa zavolajú ako reakcia na spomínané udalosti. Napríklad kódom `api.handlers.success.push(farg)` sa dá zabezpečiť, že funkcia `farg` bude zavolaná, ak server zašle odpoveď indikujúcu to, že používateľ odpovedal na všetky otázky správne a bol overený. Udalosti, na ktoré je možné registrovať funkcie sú: nová CAPTCHA, nesprávna odpoveď, úspešné vyriešenie otázky, úspešné vyriešenie celého testu, používateľ klikol na tlačidlo alternatívneho testu, používateľovi to tvralo príliš dlho, pred odoslaním a pri chybe. Všetky registrované funkcie sú volané s nezmenenou odpoveďou od servru. To umožňuje vlastným funkciám pridať ďalšiu obsluhu. Ďalej obsahuje metódy na nastavovanie textu GUI prvkov. Používa sa JavaScript len do ES5 a nepoužívajú sa žiadne prídavné knižnice typu jQuery. Tie by totiž zbytočne vytvárali ďalšiu nutnú závislosť, ako aj potrebu prenášať celú veľkosť externých knižníc. V apjScript.js sa očakáva, že sa v dokumente nachádza prázdny div element s triedou „apjCaptcha“, v ilcScript.js div element s triedou „ilcaptcha“. Do týchto prvkov bude vložené GUI.

9 Vyhodnotenie

V tejto sekcii budú stručne vyhodnotené CAPTCHA testy vytvorené v práci. Hodnotenie bude spočívať v meraní rýchlosti a úspešnosti vyriešenia človekom. Namerané rýchlosti a úspešnosť sú striktne orientačné – testy boli riešené veľmi malou vzorkou ľudí. Namerané rýchlosti predstavujú rýchlosť odpovedania na jednu otázku, nie na sadu viacerých.

Piatim ľudom bolo vysvetlené, že budú mať vyriešiť jednoduchú úlohu. Pri prvom riešení APJ CAPTCHA boli ešte uvedomení, že sa nemusia snažiť správne interpretovať celý text, ale že majú zvoliť správnu odpoveď čo najskôr. Každý človek riešil každý zo 4 typov testov 20 krát. Výsledky testovania prehľadne znázorňuje graf 30. Z grafu je vidieť, že najrýchlejšie bolo pre ľudí riešiť LayerCaptcha. Medián vyriešenia jednej otázky je približne len 4 sekundy. Kontrastne najpomalšia je neprekvapivo IQCaptcha, kde sa čas riešenia jednej otázky pohyboval medzi dvadsiatimi sekundami, až takmer dvoma minútami v zopár prípadoch. Za prekvapivý výsledok považujem rýchlosti APJ a Illucaptcha. Očakávaná bola problematickosť rýchlo dešifrovať nezrozumiteľný text a následne ho ešte k tomu zaradiť k jednej z troch kategórií. Naopak, pri Illucaptcha som očakával pre jej jednoduchosť rýchlosť na úrovni LayerCaptcha. Ľuďom zabralo prvých pár sekúnd uvedomenie si, ktoré tvary akej farby majú vlastne za úlohu spočítať. Nie je vylúčená ani jazyková bariéra, keďže Illucaptcha je v angličtine a nikto z dobrovoľníkov anglicky plynule nehovorí.

Chybovosť bola nasledovná. Na každý CAPTCHA test mal človek 20 pokusov. Najmenšou chybovosťou sa pýši LayerCaptcha: dvaja ľudia neodpovedali



Obr. 30: Rýchlosti riešení testov

správne 2 krát, dvaja raz a jeden človek skončil s počtom nesprávnych odpovedí 1,5[21]. O niečo horšia na tom bola chybovosť riešení APJ testu. Jeden človek odpovedal nesprávne raz, traja dvakrát, jeden trikrát. Znovu prekvapivo zlá bola úspešnosť Illucaptcha: 2, 2, 3, 3, 6. Očakávanie splnila IQCaptcha, kde mali traja dobrovoľníci 4 neúspešné pokusy a dvaja dva. Ak by sme ale mali brať do úvahy aj predvolený časový limit 60 sekúnd, chybovosť by bola dvojnásobná.

[21]jedna nesprávna odpoveď + raz si popýtal nový test cez tlačidlo „I don't know"

Záver

V teoretickej časti práce je predstavený úvod do CAPTCHA problematiky. Stručne sú vysvetlené príčiny potreby overovania používateľov v dnešnom internete. Práca popisuje rôzne prístupy na riešenie tohto problému, ich výhody alebo nevýhody v modernom webe. Ďalej sa práca venuje nemnoho súčasným moderným implementáciám, náznakovo zhrňuje, ako bývajú tieto CAPTCHA testy prelomené.

Objemnejšiu časť tvorí návrh a implementácia vlastných – nových CAPTCHA testov. Výsledný CAPTCHA test LayerCaptcha generovaný s použitím technológie Node.js overuje používateľov na základe prirodzenej schopnosti človeka sémanticky rozlíšiť cudzie časti obrazu. Druhý test APJ CAPTCHA napísaný v Jave a Node.js spolieha na dispozíciu gramotného človeka prečítať text s pophrehadzovanými znakmi a na zaradenie textu do troch sémanticky odlišných skupín. Python modul Illucaptcha využíva kombináciu obrázkovej a textovej úlohy, kdežto Node.js IQCaptcha overuje používateľov na základe obrázkovej logickej hádanky. Výsledkom práce sú taktiež moduly poskytujúce overovanie, kaskádové štýly a malé JavaScriptové knižnice umožňujúce pohodlné nasadenie.

Conclusions

The theoretical part presents the introduction to CAPTCHA. We briefly explain the reasons for the need of user authentication in today's Internet. This work describes various approaches to solving this problem, their advantages or disadvantages in modern web. Furthermore, the work deals with a few current modern implementations, concisely summarizes how these CAPTCHA tests are broken.

The wider part consists of design and implementation of own proposed new CAPTCHA tests. The resulting CAPTCHA LayerCaptcha implemented using Node.js technology verifies users based on the human's natural ability to semantically distinguish foreign parts of an image. The second APJ CAPTCHA test implemented in Java and in Node.js relies on a literate person's disposition to read the text with substituted characters and to classify the text into three semantically different groups. Python-based Illucaptcha uses a combination of image and text tasks, while Node.js' IQCaptcha authenticates users based on image logic puzzles. The result of this work are also modules providing authentication, cascading styles and small JavaScript libraries for convenient deployment.

A Obsah priloženého CD

doc/
: Text práce vo formáte PDF, vytvorený s použitím záväzného štýlu KI PřF UP v Olomouci pre záverečné práce, vrátane všetkých príloh, a všetky súbory potrebné pre bezproblémové vygenerovanie PDF dokumentu textu (v ZIP archíve), t.j. zdrojový text textu, vložené obrázky, apod.

src/
: Kompletné zdrojové texty implementovaných CAPTCHA testov, autentifikátory, klientské CSS a JavaScript súbory. Zdrojové kódy taktiež ukážkových stránok.

install/
: Inštalátory Node.js, PHP 5.6, PHP 7.1.30, Python 3.7.4.

readme.txt
: Inštrukcie pre nasadenie ukážkových webservrov využívajúcich implementované CAPTCHA.

Bibliografia

[1] *CAPTCHA: Telling Humans and Computers Apart Automatically*. [online] [cit. 2019-7-22]. Dostupné z http://captcha.net/

[2] *Learn About CAPTCHA: No Bots Allowed*. [online] 2019-06-01. [cit. 2019-7-22]. Dostupné z https://www.whoishostingthis.com/resources/captcha/

[3] AHMAD, A.S. El; YAN, J.; TAYARA, M. The Robustness of Google-CAPTCHAs. Newcastle University, 2011

[4] *Choosing the type of reCAPTCHA*. [online] 2019 [cit. 2019-7-22]. Dostupné z https://developers.google.com/recaptcha/docs/versions

[5] *Classes*. [online] 2019 [cit. 2019-7-22]. Dostupné z https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Classes

[6] *Making asynchronous programming easier with async and await*. [online] 2019 [cit. 2019-7-22]. Dostupné z https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Async_await

[7] PASQUALI, Sandro. *Mastering Node.js*. 2013 ISBN 9781782166320

[8] *Bézier curve*. [online] 2019. [cit. 2019-7-22] Dostupné z https://en.wikipedia.org/wiki/B%C3%A9zier_curve

[9] TATROE, Kevin; MACINTYRE, Peter; LERDORF, Rasmus. *Programming PHP*. 2013. ISBN 9781449392772

[10] *PHP Versions Stats - 2019.1 Edition*. [online]. 2019. [cit. 2019-7-22]. Dostupné z https://blog.packagist.com/php-versions-stats-2019-1-edition/

[11] *Express – Fast, unopinionated, minimalist web framework for Node.js*. [online]. 2019. [cit. 2019-7-22]. Dostupné z https://expressjs.com/

[12] FLANAGAN, David; BENJAMIN, Evans J.; *Java In A Nutshell*. 5st. 2015 ISBN 9789351108511.

[13] LUTZ, Mark. *Learning Python*. 5st. 2013. ISBN 9781449355739

[14] GRINBERG, Miguel. *Flask Web Development*. 2014. ISBN 9781449372620.

[15] GEARY, David. *Core HTML5 Canvas: Graphics, Animation, and Game Development*. 1st. Upper Saddle River, NJ, USA: Prentice Hall Press, 2012. ISBN 9780132761611.