Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

FIIT-5220-46168

Bc. Filip Šoltés

ZVÝŠENIE BEZPEČNOSTI WEBOVÉHO SYSTÉMU POUŽITÍM BIOLÓGIOU INŠPIROVANÝCH METÓD

Diplomová práca

Vedúci práce: doc. Ing. Ladislav Hudec, CSc.

máj, 2016

Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií

FIIT-5220-46168

Bc. Filip Šoltés

ZVÝŠENIE BEZPEČNOSTI WEBOVÉHO SYSTÉMU POUŽITÍM BIOLÓGIOU INŠPIROVANÝCH METÓD

Diplomová práca

Študijný program: Softvérové inžinierstvo
Študijný odbor: 9.2.5 Softvérové inžinierstvo
Miesto vypracovania: Ústav aplikovanej informatiky, FIIT STU Bratislava
Vedúci práce: doc. Ing. Ladislav Hudec, CSc.

máj, 2016

Anotácia

Slovenská technická univerzita v Bratislave
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
Študijný program: Softvérové inžinierstvo

Autor: Bc. Filip Šoltés
Diplomová práca: Zvýšenie bezpečnosti webového systému použitím biológiou inšpirovaných metód
Vedúci práce: doc. Ing. Ladislav Hudec, CSc.
máj, 2016

Veľké množstvo webových systémov v súčasnej dobe obsahuje zraniteľnosti jednoducho zneužiteľné útočníkmi, pričom následky takéhoto zneužitia môžu byť fatálne. V práci navrhujeme riešenie, ktoré zvyšuje bezpečnosť webového systému pomocou detekcie anomálií.

Pre naše riešenie využívame kombináciu umelého imunitného systému a evolučného algoritmu. Využitie tejto kombinácie sa v našom kontexte ukazuje ako vhodné, keďže útoky na webový systém môžeme chápať ako útoky patogénov na organizmus.

Navrhli sme systém ktorý zbiera a agreguje dáta webového systému pomocou špecializovaných agentov. Zozbierané dáta vyhodnocuje jedným z algoritmov umelých imunitných systémov, a to algoritmom dendritických buniek. Tento algoritmus obohacujeme o ladiacu časť v ktorej pomocou evolučného algoritmu optimalizujeme váhu dát z jednotlivých zdrojov.

Pre navrhnuté riešenie sme vytvorili implementáciu, pričom sme overili úspešnosť detekčného mechanizmu. Výsledky sme porovnali s výsledkami systému PHPIDS. V porovnaní náš systém síce dosiahol vyšší počet falošných poplachov, na základe všetkých zozbieraných dát sme ho však vyhodnotili ako lepší. Z výsledkov je viditeľný výrazný dopad ladiacej časti na celkovú úspešnosť algoritmu.

Annotation

Slovak University of Technology in Bratislava
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGY
Degree course: Software engineering

Author: Bc. Filip Šoltés
Master thesis: Improving security of a web system using biology inspired methods
Supervisor: doc. Ing. Ladislav Hudec, CSc.
2016, May

Many web systems currently contains vulnerabilities which could be easily misused by attackers, while consequences of this misuse can be fatal. In our thesis we propose a solution, that improves security of a web system by anomaly detection.

In our solution we are using combination of artificial immune system and evolutionary algorithm. Use of this combination seems proper in our context while attacks on web systems could be seen as attacks of pathogens on organism.

We have designed a system which is collecting and analysing data from web system using specialized agents. Gathered data is then evaluated with one of the artificial immune systems algorithms which is dendritic cell algorithm. We are modifying the algorithm with tuning module in which we use an evolutionary algorithm for optimization weight of data from specific sources.

For the designed solution we created an implementation and we tested relevance of the detection mechanism. We compared the results with the results of the PHPIDS system. Despite higher amount of false positives of our system we evaluated it as more suitable because of overall results. From the results we can also see visible impact of the tuning module for the overall relevance.

POĎAKOVANIE

Ďakujem vedúcemu mojej práce, doc. Ing. Ladislavovi Hudecovi, CSc., za cenné rady, konštruktívnu kritiku a motiváciu pri písaní práce. Ďakujem tiež rodine a blízkym za podporu a pomoc.

ČESTNÉ PREHLÁSENIE

Čestne prehlasujem, že záverečnú prácu som vypracoval samostatne s použitím uvedenej literatúry a na základe svojich vedomostí a znalostí.

.....

Bc. Filip Šoltés

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam algoritmov

Kapitola 1

Úvod

Pri súčasnom rozmachu používania webových systémov sa veľmi často stretávame s pokusmi o prieniky do týchto systémov. Pre ochranu používateľov, dát, alebo systémov samotných existuje niekoľko riešení, ako je použitie firewallov alebo šifrovania.

Jedným z komplexnejších spôsobov ochrany je využitie systémov na detekciu prienikov. V súčasnej dobe existuje veľa rôznych metód a prístupov pre vytvorenie systému pre detekciu prienikov. Jednou z metód je využitie biológiou inšpirovaných metód.

Cieľom našej práce je použitím znalostí získaných z existujúcich publikácií v oblasti počítačovej bezpečnosti a výpočtovej inteligencie navrhnúť, implementovať a otestovať systém, ktorý bude schopný detegovať útoky na monitorovaný webový systém. Naším cieľom je, aby bol systém použiteľný v reálnom nasadení pri čo najmenšom vyťažovaní systémových zdrojov.

Jedným z hlavných kritérií, ktoré budeme pri riešení brať do úvahy je úspešnosť detekcie prienikov. Našou prioritou je vytvoriť riešenie, ktorého výsledky budú schopné obstáť v porovnaní s výsledkami existujúcimi riešeniami, pričom budeme klásť dôraz na schopnosť detekcie útokov na neznáme zraniteľnosti.

1.1 Použité skratky

AIS z angl. *artificial immune system*, umelý imunitný systém

AI z angl. *artificial intelligence*, umelá inteligencia

API z angl. *application programming interface*, aplikačné rozhranie

CI z angl. *computational intelligence*, výpočtová inteligencia

CSA z angl. *clonal selection algorithm*, algoritmus dendritických buniek

CSRF z angl. *cross-site request forgery*, typ útoku na webové systémy

DCA z angl. *dendritic cell algorithm*, algoritmus dendritických buniek

DC dendritická bunka

DDoS z angl. *distributed denial of service*, typ útoku na webové systémy

DOM doménový objektový model

DoS z angl. *denial of service*, typ útoku na webové systémy

EA evolučný algoritmus

FN z angl. *false negative*, typ odpovede systému na detekciu prienikov

FP z angl. *false positive*, typ odpovede systému na detekciu prienikov

FPR z angl. *false positive rate*, miera TP

IDS z angl. *intrusion detection system*, systém na detekciu prienikov

JSON z angl. *javascript object notation*, formát serializácie dát

REST z angl. *representational state transfer*, webový softvérový architektonický štýl

TN z angl. *true negative*, typ odpovede systému na detekciu prienikov

TP z angl. *true positive*, typ odpovede systému na detekciu prienikov

TPR z angl. *true positive rate*, miera TP

XSS z angl. *cross site scripting*, typ útoku na webové systémy

1.2 Použité pojmy

escapovanie spôsob ošetrovania špeciálnych znakov v reťazci

token kategorizovaný blok textu

cookie malé množstvo dát, ktoré je prijaté z web stránky a uložené v prehliadači používateľa

Kapitola 2

Analýza problematiky bezpečnosti webových systémov

2.1 Bezpečnosť webových systémov

V nasledovných sekciách opíšeme problematiku bezpečnosti webových systémov. Naša analýza pozostáva z opisu oblastí potrebných pre úspešný návrh vlastného systému pre zvýšenie bezpečnosti webového systému.

V našej analýze opisujeme niekoľko najznámejších a najčastejších druhov útokov na webové systémy, následne predstavujeme systémy na detekciu prienikov a ich základné rozdelenie. Nakoniec opisujeme existujúce systémy na detekciu prienikov.

2.2 Zraniteľnosti webových systémov

Zraniteľnosť webového systému je cesta, ktorú môže útočník použiť pre zneužitie webového systému [24]. Celkové riziko zneužitia webového systému potom môžeme určiť ako kombináciu zložitosti nájdenia tejto cesty a odhadom obchodného a technického dopadu na celú organizáciu.

V nasledujúcich sekciách priblížime niektoré najznámejšie a najčastejšie zneužívané zraniteľnosti webových systémov, pričom sa zameriame na zraniteľnosti aplikačnej vrstvy.

2.2.1 Injektovanie

K injektovaniu dochádza v prípadoch, keď sú do aplikácie posielané dáta, ktoré môžu byť interpretované ako časť príkazu alebo dopytu [24]. Zraniteľné sú aplikácie využívajúce SQL, LDAP a pod.

Útok využíva prípady, kedy sa do pripravovaných príkazov alebo dopytov v aplikácii vkladajú reťazce získané priamo od používateľa a najmä prípady, kedy tieto reťazce nie sú aplikáciou upravované.

Injektovaniu je možné predísť niekoľkými spôsobmi [24]:

Parametrizácia. Ak je to možné, pri skladaní príkazov a dopytov by malo byť využívané API, ktoré zamedzuje priamy prístup k interpreteru. Túto vlastnosť má v súčasnej dobe implementovanú väčšina rámcov.

Escapovanie. V prípade, kedy nie je možné využiť parametrizáciu, je potrebné escapovať znaky, ktoré by mohli byť interpretované ako časť syntaxe jazyka príkazu alebo dopytu.

Validácia vstupov. Reťazec, ktorý je získaný od používateľa by mal prejsť kontrolou na prítomnosť znakov, ktoré v danom kontexte v reťazci nemajú zmysel.

2.2.2 Cross-Site Scripting

Cross-site scripting (ďalej len XSS) využíva zraniteľnosti webovej aplikácie ktorá umožňuje do systému vkladať a spúšťať skripty z klientskej strany [5]. Tento typ útoku je najčastejšie objavujúcim sa typom útoku na webových systémoch. Útoky XSS môžeme rozdeliť do troch hlavných kategórií:

- spôsobujúce perzistentné akcie,
- spôsobujúce neperzistentné akcie.
- založené na dokumentovom objektovom modeli (ďalej len DOM).

Prvé dva typy útokov sú spôsobené zraniteľnosťou kódu na serverovej strane, tretí je spôsobený zraniteľnosťou kódu na klientskej strane.

Pri útoku, ktorý vykonáva perzistentné akcie sa trvale mení obsah webového systému. Sú to napríklad zobrazované dáta, alebo zmeny v konfigurácii systému. Útok, ktorý spôsobí neperzistentú akciu je väčšinou zameraný na získanie existujúcich dát, alebo dočasný prístup do štandardne blokovaných častí systému. Útok založený na DOM spôsobuje zmeny v HTML kóde, ktorý je spracovávaný klientmi.

Útokom XSS je možné predísť niekoľkými spôsobmi [24]:

Escapovanie. Podobne ako v sekcii 2.2.1.

Validácia vstupov. Podobne ako v sekcii 2.2.1.

Využitie existujúcich knižníc. Existujúce riešenia sú schopné upraviť používateľský vstup pre bezpečné použitie v aplikácii, pričom sú zamerané na XSS.

Content Security Policy Vyžitím tohto štandardu je možné v prehliadači definovať umiestnenie a typ obsahu, ktorý má byť načítaný.

2.2.3 Cross-Site Request Forgery

Cross-Site Request Forgery (ďalej len CSRF) je typ útoku, pri ktorom je používateľ prinútený spustiť nechcené akcie v rámci aplikácie, v ktorej je autentifikovaný [5]. Prehliadač používateľa odosiela zraniteľnej aplikácii podvrhnutý HTTP dopyt obsahujúci informácie o aktuálnej relácii používateľa. Vďaka týmto informáciám je útočník schopný posielať podvrhnuté dopyty, ktoré sú aplikáciou považované za legitímne [24].

Predchádzať útokom CSRF je väčšinou možné vkladaním nepredvídateľného tokenu do HTTP dopytov. Token by mal byť unikátny pre reláciu a v ideálnom prípade by mal byť prenášaný v skrytom poli, keďže prenášanie ako parameter URL zvyšuje bezpečnostné riziko [24].

2.2.4 Útoky DoS

Ako dokazujú nedávne útoky na komerčné servery a poskytovateľov pripojenia, DoS útoky (z angl. *Denial of Service*) sú naliehavým problémom na internete. DoS útoky slúžia na zneprístupnenie služieb, a to akýmkoľvek spôsobom. Od vytrhnutia kábla zo servera až po využitie slabiny v aplikácii. Môže ísť o dosiahnutie limitu sieťovej karty, aplikácie, či systémových prostriedkov (CPU, pamäť, miesto na disku . . .).

Ak útok prichádza z jedného zdroja, je často pomerne jednoduché útok zastaviť napríklad blokovaním dopytov z jeho IP adresy. Ak útok prichádza z viac ako jedného zdroja, je zastavenie útoku veľmi komplexný problém. Takéto útoky nazývame DDoS (z angl. *Distributed Denial of Service*).

V prvých prípadoch ich výskytu boli tieto útoky len technickými hrami v komunite útočníkov. Keďže na vykonanie tohto typu útoku stačilo využiť niektorý z voľne šírených nástrojov, útoky začali byť vykonávané aj bežnými používateľmi.

Dôvody útokov DoS môžu byť rôzne, často ale však ide o snahu znefunkčnenie infraštruktúry organizácií s ktorými útočníci nesúhlasia, alebo o odstavenie konkurencie. V poslednej dobe sa často môžeme stretnúť aj s prípadmi, kedy útočníci vydierajú firmy hrozbami DoS útokov pričom vyžadujú výkupné.

2.2.5 Path traversal

Útok *path traversal* je zameraný na prístup k súborom a adresárom, ktoré sú umiestnené mimo koreňového adresára web servera. Útočník skúma aplikáciu a hľadá absolútne odkazy na súbory. Manipuláciou premenných pomocou reťazcov typu "../", ktoré slúžia na navigáciu v unixových súborových systémoch útočník dokáže pristupovať k súborom, ku ktorým by bežný používateľ prístup mať nemal. Tento útok môže byť vykonaný aj spustením škodlivého kódu vloženého do cesty k súboru.

Na vykonanie tohto útoku nie sú potrebné špeciálne nástroje, stačí zistiť, ktoré URL sú prístupné.

2.3 Systémy na detekciu prienikov

Systémy na detekciu prienikov (ďalej len IDS, z angl. *intrusion detection system*) sú softvérové systémy, ktoré majú za úlohu zabrániť zneužitiu počítačového systému, alebo siete [21]. Tieto systémy získavajú informácie o prostredí, v ktorom sú nasadené a vykonávajú bezpečnostnú diagnózu [10].

Z pohľadu správnosti diagnózy systém môže vrátiť niekoľko typov odpovedí.

True positive (ďalej len TP). Systém správne vyhodnotí zaznamenané dáta ako hrozbu.

False positive (ďalej len FP). Systém zaznamenané dáta vyhodnotí ako hrozbu napriek tomu, že ide o korektné správanie systému.

True negative (ďalej len TN). Systém správne vyhodnotí zaznamenané dáta ako korektné správanie systému.

False negative (ďalej len FN). Systém nesprávne vyhodnotí zaznamenané dáta ako korektné správanie, pričom ide o hrozbu.

Vyhodnotením počtu jednotlivých typov výsledkov je možné určiť úspešnosť IDS formou miery TP (ďalej len TPR), ktorá je definovaná vo vzťahu 2.1 a formou miery FP (ďalej len FPR), ktorá je definovaná vo vzťahu 2.2.

$$TPR = \frac{|TP|}{|TP| + |FN|} \tag{2.1}$$

$$FPR = \frac{|FP|}{|FP| + |TN|} \tag{2.2}$$

IDS používajú na detekciu prienikov niekoľko druhov mechanizmov, ktoré môžeme rozdeliť do dvoch hlavných skupín:

1. detekčný mechanizmus založený na príznakoch,
2. detekčný mechanizmus založený na anomáliách.

Vyššie spomenuté skupiny sú komplementárne a predstavujú úplne iné prístupy pre detekciu prienikov.

2.3.1 Detekčný mechanizmus založený na príznakoch

Detekčný mechanizmus založený na príznakoch je aplikovateľný na útoky, o ktorých je zhromaždené dostatočné množstvo údajov [10]. IDS obsahuje vo svojej databáze informácie o týchto útokoch, pričom kontroluje pokusy o tieto útoky porovnávaním nazbieraných údajov s údajmi v databáze.

Výhodou tohto detekčného mechanizmu je, že by sa v jeho výsledkoch mal nachádzať nízky počet FP a z toho vyplývajúca vysoká presnosť mechanizmu.

Nevýhodou je náročnosť získavania informácií o útokoch a taktiež fakt, že touto metódou sme schopní detegovať len známe typy útokov.

Systémy využívajúce tento detekčný mechanizmus boli v minulosti implementované hlavne pomocou logických a expertných systémov. V komerčných

riešeniach sú najčastejšie používané metódy ako porovnávanie vzoriek, Petriho siete alebo analýza stavových prechodov [10].

2.3.2 Detekčný mechanizmus založený na anomáliách

Detekčný mechanizmus založený na anomáliách pri detekcii vyhľadáva správanie, ktoré sa odlišuje od normálneho správania sa systému. Pred začiatkom používania systému je potrebné vytvoriť model normálneho správania systému, s ktorým sú potom porovnávané údaje získané z normálnej prevádzky daného systému.

Veľkou výhodou tohto detekčného mechanizmu je schopnosť detegovať aj predom neznáme typy útokov alebo typy útokov, ktoré je náročné detegovať mechanizmom založenom na príznakoch.

Nevýhodou mechanizmu je vysoký počet FP, ktorý je spôsobený náročnosťou zachytenia všetkých aspektov normálneho správania systému vo fáze učenia.

Systémy využívajúce tento detekčný mechanizmus sú často implementované pomocou expertných systémov, neurónových sietí, alebo umelých imunitných systémov [10].

2.3.3 Požiadavky efektívneho IDS

Pre vytvorenie efektívneho IDS bolo definovaných sedem požiadaviek [21]:

1. robustnosť,
2. konfigurovateľnosť,
3. rozšíriteľnosť,
4. škálovateľnosť,
5. prispôsobivosť,
6. globálna analýza,
7. efektívnosť.

Na základe vyššie uvedených požiadaviek bolo v [21] definovaných šesť základných evaluačných kritérií pre IDS.

Distribuovanosť. Podporuje robustnosť, keďže zlyhanie jedného detekčného procesu nemá vplyv na funkčnosť celého IDS. Distribuované systémy sú jednoducho rozšíriteľné a zároveň škálovateľné. Konfigurovateľnosť systému je jednoduchšia, keďže vie vytvoriť konfigurácie pre jednotlivé uzly distribuovaného systému.

Schopnosť samostatnej organizácie. Bez externého spravovania alebo údržby dokáže IDS detegovať predom neznáme alebo distribuované prieniky. Takéto systémy sú vysoko prispôsobivé, keďže nie je potrebné manuálne obnovovať vnútornú databázu s opismi jednotlivých prienikov.

Nízka náročnosť na zdroje. Systém by nemal byť záťažou pre uzol, na ktorom je spustený, keďže monitorovanie spravidla nie je primárna funkcia uzla.

Viacvrstvovosť. Pre zvýšenie robustnosti by IDS mal byť organizovaný viacvrstvovo, keďže výpadok jednej vrstvy nemusí mať vplyv na funkčnosť celého systému. Viacvrstvovosť môžeme dosiahnuť zaznamenávaním údajov viacerých vrstiev systému.

Rozmanitosť. Rozmanité druhy detekčných procesov rozdelených do rôznych hostov spomalí útok, ktorý už kompromitoval jeden, alebo viacero hostov.

Nezávislosť na komponentoch. IDS, ktorý je nezávislý na komponentoch, je schopný výmeny jednotlivých komponentov bez dosahu na funkčnosť celého systému.

2.4 Existujúce riešenia

V súčasnej dobe existuje veľké množstvo rôznych komerčných, alebo nekomerčných riešení IDS. V nasledovných sekciách predstavíme dva populárne, voľne dostupné IDS - *Snort* a *PHPIDS*.

2.4.1 Snort

Snort je populárny systém pre detekciu prienikov obsahujúci detekčný mechanizmus založený na príznakoch. Je široko využívaný najmä na inšpekciu sieťovej prevádzky a je hodnotený ako jeden z najkvalitnejších dostupných systémov [6].

Snort pre detekciu využíva pravidlá, ktoré sú uložené v textových súboroch jednoducho modifikovateľných pomocou textového editora. Po každom štarte sú pravidlá načítané a pričom je vytvorená interná dátová štruktúra určená na aplikovanie pravidiel pre zachytené dáta. V systéme je veľké množstvo preddefinovaných pravidiel, ktoré detegujú pokusy o prieniky.

Snort je logicky rozdelený do niekoľých komponentov [6]:

- dekóder paketov,
- preprocesory
- detekčný mechanizmus
- systém pre logovanie a upozorňovanie
- výstupné moduly

Tieto komponenty spolupracujú pri detekcii konkrétnych útokov a pri generovaní výstupu vo vyžadovanom formáte.

Zdrojové kódy sú publikované pod GNU GPL licenciou.

2.4.2 PHPIDS

PHPIDS je voľne šíriteľný IDS určený pre aplikácie založené na jazyku PHP [1]. Pomocou skúmania vstupných premenných v POST a GET dopytoch a v HTTP cookie dokáže PHPIDS detegovať väčšinu konvenčných útokov na webové systémy ako XSS, SQL injektovanie, DoS, path traversal a pod.

PHPIDS primárne využíva detekčnú metódu založenú na príznakoch, a teda pre detekciu využíva preddefinované vzorky. Systém však obsahuje komponenty pre hĺbkovú analýzu reťazcov a metrík na vstupe pomocou ktorej je

možné detegovať aj neznáme útoky [1]. HĺbkovÁ analýza je však pomerne jednoduchá a pozostáva z výpočtu pomeru počtu znakov, ktoré sa môžu vyskytovať v slovách (alfanumerické znaky, interpunčné znamienka) k počtu ostatných znakov v reťazci. Systém umožňuje viacero foriem výstupu ako napríklad relačná databáza, log súbor alebo odoslanie emailu.

Ako názov napovedá, aplikácia je implementovaná v PHP a je licencovaná pod GNU LGPL licenciou.

Kapitola 3

Analýza vybraných prístupov výpočtove inteligencie

3.1 Prístupy výpočtovej inteligencie

Pre detekciu prienikov je často používaná výpočtová inteligencia [33] (ďalej len CI, z angl. *computer intelligence*). Od umelej inteligencie (ďalej len AI, z angl. *artificial intelligence*) sa odlišuje hlavne tým, že nepoužíva symbolickú reprezentáciu znalostí ako AI, ale pracuje výhradne s nízkoúrovňovými numerickými reprezentáciami údajov [11]. V kontexte IDS sa často používa niekoľko prístupov CI, a to najmä:

Umelé neurónové siete Množiny jednotiek zvaných *neuróny*, ktoré sú priamo spojené a usporiadané do určitej topológie, pričom sú schopné učiť sa pomocou príkladov.

Fuzzy množiny Množiny, v ktorých nie je príslušnosť prvku určovaná binárne, ale hodnotou z intervalu $< 0, 1 >$.

Evolučné výpočty Využitie princípov teórie evolúcie a prirodzeného výberu Ch. Darwina a teórie dedičnosti G. Mendela pre automatizované riešenie problémov [22]

Umelé imunitné systémy Využitie abstrakcií biologického imunitného systému pre detekciu abnormálneho správania monitorovaného systému [21].

Inteligencia roja Metóda, ktorá je podobná evolučným výpočtom, ale namiesto vytvárania nových prvkov upravuje existujúce na základe vzájomnej komunikácie.

Kombinácia viacerých prístupov Časté alternatívy sú napr. evolučné výpočty s fuzzy množinami, umelé neurónové siete s fuzzy množinami či iné.

V nasledujúcich sekciách detailnejšie analyzujeme dva z prístupov CI a to evolučné výpočty a umelé imunitné systémy.

3.2 Umelý imunitný systém

3.2.1 Základy biologického imunitného systému

V medicíne pojem imunitný systém predstavuje obranný mechanizmus organizmu, ktorý slúži na ochranu proti chorobám. Štrukturálne imunitný systém pozostáva z buniek, molekúl, tkanív, orgánov a obehového systému [30]. Koordinovaná reakcia všetkých častí imunitného systému na prítomnosť cudzích látok, nazývaných *patogény*, je označovaná ako imunitná reakcia [9].

Imunitný systém je často rozdeľovaný do dvoch rozdielnych, ale prepojených podsystémov [30]:

1. špecifický imunitný systém,
2. nešpecifický imunitný systém.

Hlavnou charakteristikou špecifického alebo získaného imunitného systému je rozoznanie patogénov, čo má za následok vytvorenie dlhodobej pamäte pre konkrétny patogén. Špecifický imunitný systém nie je statický, ale vyvíja sa pomocou interných a externých vnemov a je organizovaný hlavne okolo *T buniek* a *B buniek*.

T bunky a B bunky patria medzi biele krvinky nazývané *lymfocyty* a sú dôležitou súčasťou imunitného systému, pričom podľa ich typu zastávajú rôzne funkcie. Väčšinu populácie lymfocytov predstavujú *B bunky* a *T bunky*, ktoré rozoznávajú patogény pomocou *antigénu*, čo je tvar molekuly na jej povrchu [9].

B bunky slúžia na rozoznanie konkrétneho antigénu a tvorbu protilátok, proteínov, ktoré dokážu rozoznať patogény naviazaním na antigén [9].

T bunky na základe ich druhu buď vysielajú chemické inštrukcie nazývané *cytokíny* do ostatných častí imunitného systému, rozoznávajú a ničia patogény priamo, alebo spolupracujú pri tvorbe protilátok [9].

Nešpecifický alebo vrodený imunitný systém je charakterizovaný tromi úlohami: obrana v skorých štádiách infekcie pomocou nešpecifického rozoznania patogénu, vyvolanie špecifického imunitného systému a výber špecifickej odozvy [30].

Nešpecifický imunitný systém je fixný a geneticky určený, pričom je organizovaný okolo niekoľkých druhov buniek a to najmä, *NK bunky* a *dendritické bunky*. NK bunky (z angl. *natural killer*) patria medzi lymfocyty a slúžia na likvidáciu nádorových buniek, či buniek napadnutých vírusmi.

Funkcia dendritických buniek bola objavená až v nedávnej dobe napriek tomu, že ich existencia bola objavená už v roku 1868. Dendritické bunky slúžia na zbieranie, spracovávanie a poskytovanie antigénov T bunkám [14].

3.2.2 Algoritmické abstrakcie imunitného systému

V oblasti imunitných systémov existuje niekoľko modelov, ktoré sa zameriavajú na pochopenie biologických procesov na vyššej úrovni.

Jedným z modelov, ktorý je aj v súčasnej dobe široko akceptovaný odborníkmi je model *self/nonself diskriminácie* [30]. Model akceptuje existenciu len elementov, ktoré sú označené ako *self*, teda vlastné a snaží sa eliminovať elementy označené ako *nonself*, teda cudzie [31].

Ďalším modelom je *teória nebezpečenstva*, ktorá namiesto konceptu *self/nonself* využiva koncept bezpečnosti a nebezpečnosti. Táto zmena sa nemusí zdať ako podstatná, rieši však jeden z problémov *self/nonself diskriminácie*, ktorým je fakt, že v tele existujú prvky, ktoré sú cudzorodé a imunitný systém ich napriek tomu neeliminuje. *Teória nebezpečenstva* pre detekciu anomálie používa signály, ktoré bunka vysiela pri neprirodzenej smrti.

Pre tieto modely existuje veľké množstvo jednotlivých prístupov a algoritmických abstrakcií [21][29][20].

Pri vytváraní jednotlivých abstrakcií bol v počiatkoch uvažovaný len špecifický imunitný systém, prípadne boli v algoritme časti špecifického a nešpecifického systému striktne oddelené. Tieto algoritmy sa nazývajú algoritmy AIS prvej generácie [30].

Neskoršie snahy o abstrakciu imunitného systémy pristupujú k špecifickému a nešpecifickému imunitnému systému ako k dvom úzko prepojeným častiam. Tieto algoritmy sa nazývajú algoritmy AIS druhej generácie [30].

V nasledujúcich sekciách opíšeme dva zo základných algoritmov AIS prvej generácie, ktorými sú algoritmus negatívnej selekcie a algoritmus klonálnej selekcie a jeden z algoritmov AIS druhej generácie ktorým je algoritmus dendritických buniek.

3.2.3 Algoritmus negatívnej selekcie

Algoritmus implementuje model *self/nonself diskriminácie* a jeho priebeh je možné rozdeliť do troch častí [21]:

1. definovanie *self*,
2. generovanie detektorov,
3. monitorovanie výskytu anomálií.

V prvej fáze je definované normálne správanie monitorovaného systému, pričom normálne vzory správania sú označené ako *self*.

V druhej fáze je vygenerovaná množina náhodných detektorov, ktoré sú porovnané s množinou *self* vytvorenou v prvej fáze. Ak detektor deteguje množinu *self*, je odstránený a na jeho miesto je vygenerovaný nový detektor. Proces sa opakuje až do stavu, kedy je vytvorený žiadaný počet detektorov, ktoré nedetegujú *self*.

V poslednej fáze je monitorovaný aktuálny stav systému voči vytvoreným detektorom. Ak nejaký z detektorov vyhovuje stavu systému, môžeme predpokladať že nejde o *self*, a teda že nastala anomália.

Množina *self* ako aj množina detektorov je spravidla reprezentovaná binárnymi reťazcami, ktoré reprezentujú model vytvorený nad monitorovaným systémom, pričom metód detegovania je niekoľko. Je možné použiť hammingovu vzdialenosť alebo počet rovnakých za sebou idúcich bitov. Je však aj množstvo iných, komplexnejších spôsobov. Kritickou časťou je určenie hranice, kedy je stav systému označený ako anomália. Táto hodnota má priamy dosah na počet FN a FP.

Jednou z veľkých nevýhod algoritmu je veľká výpočtová náročnosť. Generovanie potrebného počtu detektorov pre netriviálny systém je tak časovo náročné, že algoritmus nie je v reálnom prostredí použiteľný [21], neustále však vznikajú riešenia, ktoré sa pokúšajú proces generovania optimalizovať [31].

Ďalším problémom algoritmu je náročnosť presného zachytenia korektného správania systému do množiny *self*.

Výhodou algoritmu je jeho jednoduchosť a tiež možnosť distribuovania detektorov medzi rôznymi systémami.

3.2.4 Algoritmus klonálnej selekcie

Algoritmus klonálnej selekcie (ďalej len CSA, z angl. *clonal selection algorithm*) je populačný algoritmus, ktorý je založený na *self/nonself diskriminácii*. V CSA je každý detektor a antigén reprezentovaný množinou atribútov, pričom reprezentácia môže byť rôzna (binárne kódovanie, n-rozmerné body a pod.).

Generovanie jednotlivých detektorov prebieha procesom, ktorý je podobný evolučnému algoritmu.

Na začiatku je vytvorená náhodná množina detektorov. Následne je určená spriaznenosť každého detektora s antigénmi, pričom na základe týchto hodnôt sú jednotlivé protilátky naklonované. Počet klonov je priamo úmerný spriaznenosti. V ďalšej fáze sú detektory s najhoršou spriaznenosťou vymenené za nové, náhodne vygenerované detektory [9].

Algoritmus končí, ak množina detektorov dosiahne požadovanú veľkosť.

Ako môžeme vidieť, princíp je podobný algoritmu negatívnej selekcie, detektory však nie sú overované voči množine *self*, ale voči množine antigénov.

3.2.5 Algoritmus dendritických buniek

Algoritmus dendritických buniek (ďalej len DCA, z angl. *dendritic cell algorithm*) je populačný algoritmus, ktorý využíva model teórie nebezpečenstva a koncept bezpečnej/nebezpečnej udalosti docieľuje pomocou spracovania signálov.

Pre spracovanie jednotlivých signálov využíva abstrakciu nad dendritickými bunkami (ďalej len DC). Životný cyklus týchto buniek sa skladá z troch stavov[9]:

1. **nezrelá** - bunka zbiera vnemy a signály z tela,
2. **polo-zrelá** - bunka potláča reakciu imunitného systému,
3. **zrelá** - bunka aktivuje reakciu imunitného systému.

DC je citlivá na tri druhy signálov[12], ktoré sú vytvárané vo vláknach a bunkách.

PAMP signál. Metrika, ktorá zvyšuje svoju hodnotu pri prítomnosti anomálneho správania. Je to presvedčivý indikátor anomálie a je väčšinou prítomný pri udalostiach, ktoré určite dokážu poškodiť systém [12].

Signál nebezpečenstva. Metrika indikuje potenciálnu abnormalitu. Zvyšuje svoju hodnotu pri pozorovaní abnormálneho správania systému [12].

Signál bezpečia. Metrika zvyšuje svoju hodnotu pri pozorovaní normálneho správania. Je presvedčivým indikátorm normálneho, predpovedateľného, stabilného správania systému [12].

Vytvorením sumy metrík jednotlivých signálov bunka mení svoj stav a buď potláča, alebo aktivuje reakciu imunitného systému. Prahová hodnota počtu signálov potrebných pre zmenu stavu je parametrizovateľná. Celý priebeh algoritmu je zhrnutý v algoritme 1.

Keďže DCA je stochastický algoritmus, je veľmi obtiažne sledovanie priebehu algoritmu a evaulovať jednotlivé časti. Pre tento dôvod bola vytvorená deterministická verzia algoritmu s názvom deterministický algoritmus dendritických buniek (ďalej len dDCA, z angl. *deterministic dendritic cell algorithm*).

Správanie dDCA je predpovedateľné a systém je možné sledovať na úrovni jednotlivých antigénov. Bolo ukázané, že výsledky algoritmu neboli zmenou na deterministickú verziu porušené [16].

Algoritmus DCA, resp. dDCA ukazuje výborné výsledky a to najmä vďaka nízkym nárokom na CPU, nízkym počtom FP. Algoritmus tiež nepotrebuje rozsiahle trénovanie [17]. Vďaka tomu bolo pomocou algoritmu vytvorených

niekoľko riešení ako autormi algoritmu [15][18], tak aj mnohými ďalšími výskumníkmi [7][26].

Vstup : S = množina dát, ktoré majú byť označené ako bezpečné alebo nebezpečné
Výstup: D = množina dát označených ako bezpečné alebo nebezpečné

```
Vytvorenie prvotnej populácie DC D;
Vytvorenie množiny, ktorá obsahuje migrované DC M;
foreach prvky v S do
    Vybranie náhodnej množiny P z množiny D;
    foreach prvky v P do
        Úprava koncentrácie signálov a cytokínov;
        if koncentrácia kostimulárnych molekúl > hranica then
            Migrácia D do M a vytváranie nových DC;
        end
    end
end
foreach prvky v M do
    if |polo-zrelé cytokíny| > |zrelé cytokíny| then
        Nastavenie DC ako polo-zrelej;
    else
        Nastavenie DC ako zrelej;
    end
end
foreach prvky v S do
    Určenie počtu výskytov prvku v zrelých a polo-zrelých DC;
    if výskyt v polo-zrelých > výskyt v zrelých then
        Označenie prvku ako bezpečného;
    else
        Označenie prvku ako nebezpečného;
    end
    Pridanie prvku do M
end
```

Algoritmus 1: Pseudokód algoritmu dendritických buniek [17]

3.3 Evolučné algoritmy

Myšlienka pre použitie darwinových princípov pre automatizované riešenie problémov vznikla v 50-tych rokoch 20. storočia a v priebehu desiatich rokov postupne vznikli štyri rôzne prístupy implementácie evolúcie pri výpočtoch [32][28].

Prvé dve paradigmy pochádzajú z USA, konkrétne *evolučné programovanie*, ktoré ako prvý použil Lawrence J. Fogel a *genetické algoritmy* predstavené Johnom Henry Hollandom [32][28]. V Nemecku bola predstavená paradigma *evolučné stratégie* [32][28], ktorej autormi boli Ingo Rechenberg and Hans-Paul Schwefel. V 90-tych rokoch 20. storočia sa objavila štvrtá paradigma s názvom *genetické programovanie* [32][28].

Implementácia evolučných mechanizmov sa nazýva evolučný algoritmus.

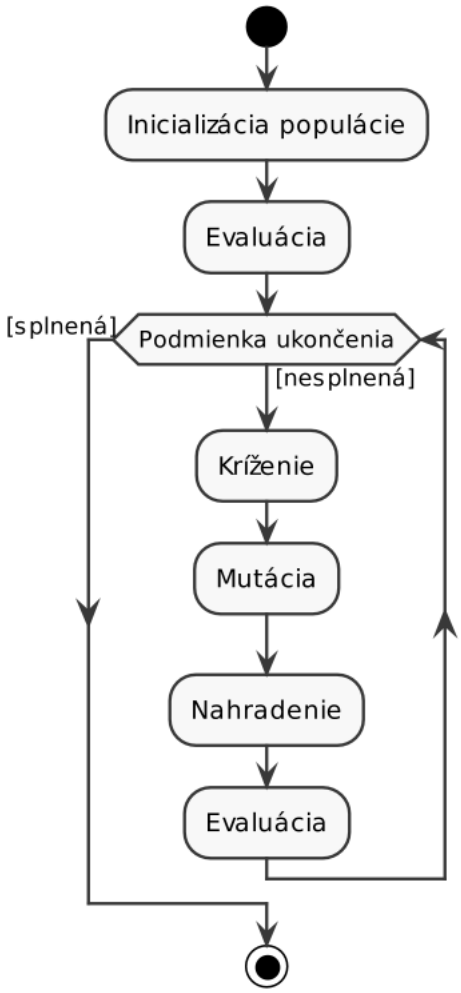
Evolučné algoritmy sú rozsiahlou triedou pravdepodobnostných populačných prehľadávacích algoritmov. Sú vhodné hlavne v prípadoch, keď je náročné nájsť správne riešenie štandardnými metódami a ako výsledok je akceptované aj riešenie, ktoré nie je optimálne, ale svojou kvalitou sa k optimálnemu riešeniu približuje.

Nutnou podmienkou úlohy, ktorá by mala byť riešená pomocou evolučných algoritmov je, aby jej riešenie bolo *rozložiteľné* a *ohodnotiteľné* [22][28]. Musíme byť teda schopní každé riešenie dekomponovať na postupnosť atribútov. Na základe hodnôt atribútov je nutné, aby bolo možné jednotlivé riešenia ohodnotiť.

Jedno konkrétne riešenie sa v evolučnom algoritme nazýva jedinec. Jedince, ktoré sú algoritmom skúmané v určitom čase sa nazývajú populácia.

3.3.1 Štruktúra evolučného algoritmu

Keďže evolučné algoritmy sú podmnožinou prehľadávacích algoritmov, ich štruktúra vychádza zo štruktúry prehľadávacích algoritmov pričom ale je možné ju konkretizovať a doplniť o časti charakteristické pre evolučný algoritmus. Príklad schémy evolučného algoritmu je možné vidieť na Obr. 3.1.



Obr. 3.1: Schéma štruktúry evolučného algoritmu.

Algoritmus, ako je zrejmé z uvedenej definície prebieha v cykle a v každej iterácii je vytvorená nová množina jedincov. Množina jedincov vytvorená v jednej iterácii je označovaná ako *generácia*.

Na začiatku evolučného algoritmu je potrebné vykonať inicializáciu, teda vytvoriť prvotnú populáciu jedincov. Táto potom tvorí základ pre vykonávanie algoritmu a je tiež označovaná ako populácia nultej generácie. Táto môže byť generovaná náhodne, alebo tak, aby čo najlepšie pokrývala množinu všetkých možných jedincov.

Po vytvorení jedincov určitej generácie je potrebné týchto jedincov ohodnotiť na základe vhodnosti z hľadiska riešenej úlohy. Táto časť algoritmu sa nazýva evaluácia, alebo *ohodnotenie* [22][28]. Na základe množiny hodnôt na

výstupe tejto časti je možné pokračovať v algoritme.

Pred tým, ako sa pristúpi k vytváraniu nových generácií je potrebné overiť podmienku pre ukončenie algoritmu. Podmienkou môže byť miera priblíženia sa k hľadanému riešeniu či prekročenie daného počtu iterácií.

Ak podmienka splnená nie je, algoritmus pokračuje vytváraním novej generácie, ktoré sa vykonáva pomocou operácií inšpirovaných teóriou evolúcie a prirodzeného výberu Charlesa Darwina a teóriou dedičnosti Gregora Mendela. Nové jedince sú tvorené reprodukciou medzi jedincami aktuálnej generácie. Reprodukcia začína *selekciou*, teda výberom najvhodnejších rodičov na základe hodnoty vhodnosti získanej ohodnotením. Po selekcii sú aplikované *genetické operátory*, ako *mutácia* či *kríženie*, pomocou ktorých sú určené hodnoty atribútov nového jedinca na základe atribútov rodičov.

Po vzniknutí nových jedincov je aplikovaná posledná časť algoritmu nazývaná *nahradenie* [22][28], ktorej úlohou je z existujúcich jedincov vytvoriť novú populáciu, ktorá bude populáciou v novej generácii.

3.3.2 Vhodnosť jedinca

Ako už bolo uvedené, vhodnosť jedinca je hodnota, ktorá určuje mieru priblíženia riešenia, ktoré reprezentuje jedinec k optimálnemu riešeniu úlohy. Vhodnosť jedincov je určovaná pomocou *fitness funkcie*.

Úlohou evolučného algoritmu je teda nájsť jedinca, ktorý dosahuje maximum na osi reprezentovanej hodnotou vhodnosti. Aby sa dala žiadaná hodnota efektívne nájsť, je potrebné, aby plocha vhodnosti mala dve vlastnosti. Mala by dosahovať čo najmenší počet lokálnych maxím. Pri hľadaní maxima totiž algoritmus môže chybne predpokladať, že sa približovaním k lokálnemu extrému blíži k optimálnemu riešeniu. Treba však zdôrazniť, že toto je vlastnosť problému a nie samotného evolučného algoritmu.

Plocha vhodnosti by tiež nemala obsahovať rozsiahle plochy s rovnakou hodnotou vhodnosti, keďže takého plochy sťažujú hľadanie smeru ďalšieho postupu hľadania.

Existuje viacero spôsobov prístupov pre voľbu fitness funkcie, pričom je vždy nutné postupovať na základe konkrétneho problému. Fitness funkcia môže byť aj kombináciou viacerých funkcií vyrobených rôznymi spôsobmi. Takýmito

spôsobmi môže byť napr. *aproximácia* či *súťaž*, kedy sa vhodnosť neurčuje globálne, ale priamym porovnávaním atribútov jednotlivých jedincov.

3.3.3 Selekcia

Selekciou sa v kontexte evolučných algoritmov rozumie výber jedincov vhodných pre reprodukciu. Pomocou selekcie je teda z populácie všetkých jedincov vybraná množina rodičov. Čím je jedinec vhodnejší, tým má väčšiu šancu byť vybraný do množiny rodičov a teda sa zapojiť do reprodukcie. Keďže množina rodičov je spravidla menšia ako množina všetkých jedincov, najvhodnejšie jedince sú často do reprodukcie vyberané niekoľkokrát.

Počas selekcie je v určitých prípadoch potrebné pracovať s inou reprezentáciou vhodnosti, preto je treba hodnoty zmeniť len pre účel selekcie. Tento proces sa nazýva *premapovanie vhodnosti* [22][28] (angl. *fitness remapping*).

Proces výberu jedincov z aktuálnej populácie sa nazýva *vzorkovanie populácie*. Výber jedincov do množiny rodičov sa vykonáva na základe pravdepodobnosti, ktorá môže byť statická, alebo dynamická, pričom sa selekčné metódy pre tieto alternatívy líšia.

Podľa toho, či sa pravdepodobnosť výberu jedinca do množiny rodičov určuje pred alebo počas začatia procesu vzorkovania, rozlišujeme vzorkovanie s *explicitnou* a vzorkovanie s *implicitnou* pravdepodobnosťou [22][28].

Vzorkovanie s explicitnou pravdepodobnosťou

Vzorkovanie s explicitnou pravdepodobnosťou je realizované pomocou metódy podobnej rulete. Počet políčok na rulete je rovný počtu jedincov v populácii, pričom veľkosť jednotlivých políčok nie je rovnaká, ale určená na základe pravdepodobnosti výberu jedinca. Po zatočení rulety je do množiny rodičov vybraný jedinec, na ktorého ukázal jazýček rulety.

Vzorkovanie s implicitnou pravdepodobnosťou

Do vzorkovania s implicitnou pravdepodobnosťou patria skupiny metód *turnaj, orezanie* a *náhodný výber*.

Turnaj Výber do množiny rodičov realizovaný na základe súťaže medzi jednotlivými jedincami. Pri tejto metóde nie je potrebné triedenie na základe vhodnosti.

Orezanie Na začiatku sú jedince zoradené podľa vhodnosti, pričom je vyradených n najhorších jedincov. Množina rodičov je potom naplnená zostávajúcimi jedincami, pričom výber je realizovaný s rovnakou pravdepodobnosťou pre všetky jedince.

Náhodný výber Výber do množiny rodičov je realizovaný zo všetkých jedincov, pričom každý jedinec má rovnakú pravdepodobnosť na výber.

3.3.4 Genetické operátory

Na vytváranie nových bodov na ploche vhodnosti sa používajú *genetické operátory*. Tieto sú aplikované na jedincov z množiny rodičov vzniknutej v procese selekcie. Cieľom genetických operátorov je vytvoriť jedinca odlišného od jedincov vzniknutých v predchádzajúcich iteráciách. Na tieto úlohy sa rozlišujú tri druhy operátorov - *asexuálne operátory, sexuálne operátory, panmiktické operátory* [2][28].

Asexuálne operátory V biologickom kontexte sa asexuálne operátory nazývajú mutačné operátory, keďže vzniknutý jedinec má len jedného rodiča.

Sexuálne operátory V biologickom kontexte sa sexuálne operátory nazývajú operátory kríženia. Vzniknutý jedinec je tvorený rekombináciou dvoch rodičov.

Panmiktické operátory Pri použití panmiktických operátorov sú nové jedince tvorené viac ako dvoma rodičmi. Tieto operátory sú v reálnych implementáciách používané najmenej.

Najčastejšie používaná je kombinácia sexuálneho a asexuálneho operátora.

3.4 Využitie CI v kontexte IDS

Existuje množstvo publikácií zameriavajúcich sa na využitie metód CI v oblasti počítačovej bezpečnosti. Jedným z riešení, ktoré kombinuje využitie IDS a CI, je systém *WCIS*[8].

WCIS je systém zameraný na detekciu neznámych útokov a je navrhnutý ako AIS doplnený o klasifikáciu jednotlivých dopytov. Táto klasifikácia má napomôcť administrátorovi poskytnutím dodatočných dát o vzniknutej udalosti [8] a je vykonávaná pomocou analýzy zadaných *URI*.

V systéme bolo definovaných šesť typov útokov, pre ktoré systém vytvoril senzory využitím normálneho a útočného datasetu. Pre jednotlivé senzory bolo potrebné trénovanie a dozretie, čo bolo docielené typickým životným cyklom algoritmu negatívnej selekcie. Pre efektívne fungovanie algoritmu bola vykonaná abstrakcia nad dátami, pričom boli zohľadnené metriky ako počet parametrov, dĺžka *URI* a pod.

Počas trénovacej fázy boli vytvorené senzory overované voči útočnému datasetu. Jednotlivým záznamom v datasete bola priradená jedna zo šiestich kategórií spomenutých vyššie. Ak senzor detegoval konkrétny útok, bol mu určený typ na základe typu detekovaného útoku. Po určení typu jednotlivých senzorov bol využitý opačný prístup, kedy jednotlivé senzory určovali typy útoku v datasete. Vytvorené dáta však boli použité len pre evauláciu jednotlivých skupín senzorov.

V ďalšej fáze bol využitý genetický algoritmus, ktorého cieľom bola optimalizácia detekčnej schopnosti jednotlivých senzorov. Jedincami v algoritme boli samotné senzory, pričom ich *fitness* bola určená ako schopnosť detegovať útoky typu priradeného v trénovacej fáze. Na jedince boli aplikované štandardné operátory evolučných algoritmov: selekcia, kríženie, mutácia a nahradenie.

Pri experimentálnom overení autori dosiahli vysokú mieru presnosti pre väčšinu typov útokov, ktoré obsahoval testovací dataset. *WCIS* bol schopný detegovať útočné dopyty bez toho, aby nesprávne upozorňoval na normálnu prevádzku. Pri teste s neznámymi dátami systém nevytvoril žiadne FP upozornenia. Nutnou podmienkou správneho fungovania algoritmu je však správne natrénovanie na normálnom datasete [8]. Trénovanie systému prebieha staticky pred nasadením a preto riešenie nedokáže pružne reagovať na zmeny v normálnom fungovaní monitorovaného systému.

Ako možné zlepšenia autori identifikovali rozšírenie vlastností vstupných dát, ktoré sú analyzované, najmä hlavičky HTTP dopytov. Ako ďalšie zlepšenie navrhli úpravu evolučného algoritmu, ktorý spôsoboval znižovanie jedinečnosti vytváraných senzorov [8].

Kapitola 4

Návrh vlastného systému na detekciu prienikov

4.1 Špecifikácia navrhovaného riešenia

Naším cieľom je navrhnúť systém, ktorý bude pomocou výpočtovej inteligencie detegovať anomálie v stave webového systému. Chceme vytvoriť komplexné riešenie, ktoré bude pracovať na aplikačnej úrovni a bude využiteľné a nasaditeľné na reálnych systémoch. Týmto máme na mysli najmä funkčnosť v reálnom čase pri množstve dát vytvorených systémov reálnej veľkosti. Ďalším cieľom je zamedzenie zásadných bezpečnostných chýb pri vytváraní architektúry.

Z hľadiska monitorovaného systému je veľmi dôležitá minimalizáciu zaťaženia zdrojov. Posledným vytýčeným cieľom je čo najväčšia autonómnosť systému, a teda minimalizácia potrebnej interakcie používateľa.

4.2 Vysokoúrovňový návrh riešenia

Na základe analyzovaných publikácií sme sa rozhodli navrhnúť vlastný IDS s detekčným mechanizmom založeným na anomáliách. Berúc do úvahy odporúčania v [21] sme sa rozhodli pre viacvrstvovú klient-server architektúru systému.

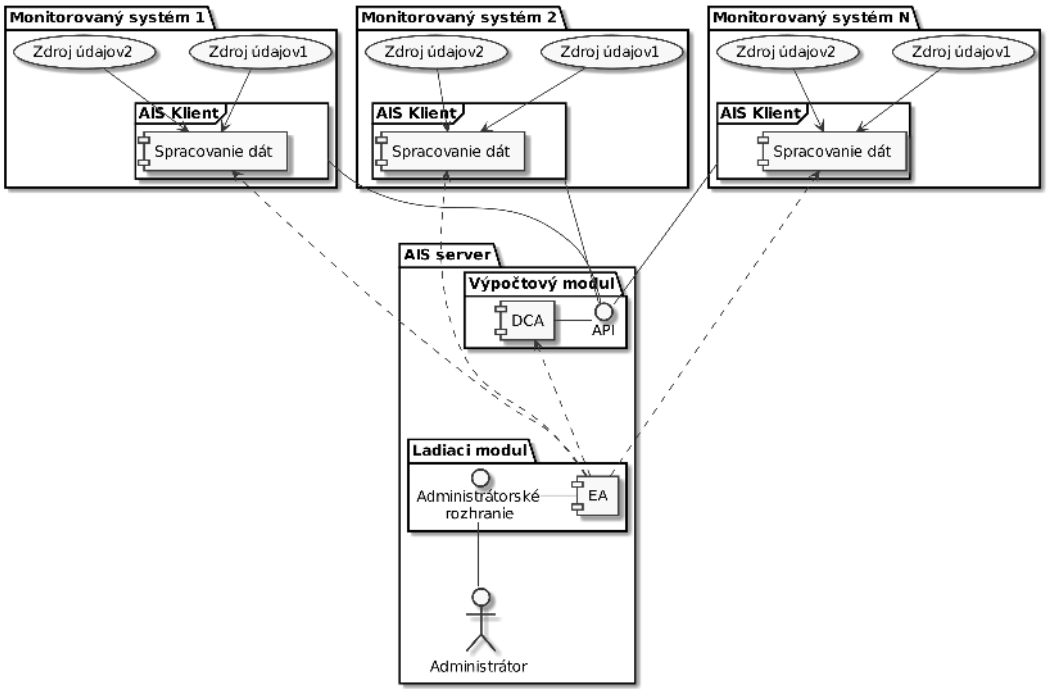
Diagram navrhovaného systému je zobrazený na obrázku 4.1. Systém je rozdelený na centrálnu serverovú časť a klientskú časť skladajúcu sa z jednotlivých agentov nasadených na monitorovaných systémoch.

Klientská časť slúži na zber a agregáciu dát, ktoré sú následne zasielané do serverovej časti. Návrh klientskej časti systému podrobnejšie opisujeme v sekcii 4.3.

Serverová časť systému slúži na detekciu anomálií spracovávaním nazbieraných dát. Spracovávanie dát je vykonávané využitím dvoch prístupov výpočtovej inteligencie:

- umelého imunitného systému,
- evolučného algoritmu.

Umelý imunitný systém slúži pre samotnú detekciu anomálií a evolučný algoritmus využívame na ladenie aplikácie. Návrh serverovej časti systému podrobnejšie opisujeme v sekcii 4.4.



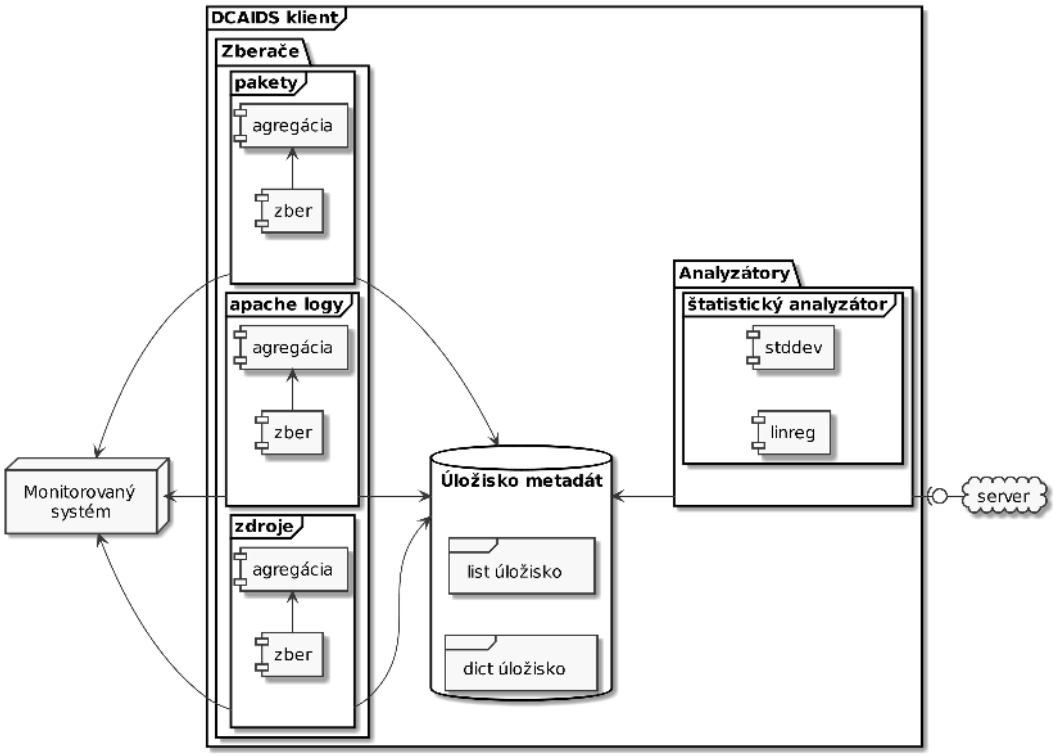
Obr. 4.1: Diagram architektúry navrhovaného systému.

4.3 Klientská časť systému

Ako už bolo spomenuté v kapitole 4.2, klientská časť aplikácie slúži najmä na zber a agregáciu dát. Tieto sú spracovávané do formy signálov a antigénov, vhodnej pre dDCA v serverovej časti, ktoré pomenúvame spoločným názvom *vzorky*. Dôležitým kritériom pri návrhu klientskej časti je čo najmenšie zaťaženie monitorovaného systému.

Zohľadnením definovaných požiadaviek sme sa rozhodli pre architektúru zobrazenú na obrázku 4.2. Návrh pracuje s troma hlavnými komponentmi:

1. komponent pre načítavanie metadát,
2. úložisko metadát,
3. komponent pre analýzu metadát.



Obr. 4.2: Diagram architektúry klientskej časti navrhovaného systému.

Pri návrhu aplikácie sme sa zamerali na čo najväčšiu modularitu a to najmä kvôli potrebe ohodnotenia jednotlivých častí riešenia. Modularita je tiež dôležitá pre jednoduchú rozšíriteľnosť aplikácie.

Toto bolo docielené využitím špecializovaných agentov pre jednotlivé úlohy, pričom sme so zohľadnením opísanej architektúry definovali dva hlavné druhy agentov:

1. zberače,
2. analyzátory.

Funkcionalita systému nie je závislá od počtu ani od konkrétnych typov agentov, nutná podmienka je len existencia aspoň jedného agenta pre každú kategóriu.

V nasledovných sekciách bližšie uvedieme princíp fungovania jednotlivých častí.

4.3.1 Načítavanie metadát

Komponent pre načítavanie metadát sa skladá aspoň z jedného agentu typu *zberač*.

Každý z agentov je napojený na jeden konkrétny zdroj údajov, z ktorého načítava surové dáta. Dáta sú následne agregované do jedného z formátov akceptovaných úložiskom metadát opísaného v sekcii 4.3.2. Po agregácii údajov sú tieto do úložiska vložené.

Cieľom agregácie surových dát je zachytenie čo najpresnejšieho obrazu aktuálneho stavu systému pomocou dát konkrétneho zdroja. Nad dátami je pritom vytvorených niekoľko abstrakcií pre efektívnejšie automatické spracovanie analyzátormi bližšie opísanými v sekcii 4.3.3.

Špecifikovali sme tri zdroje údajov:

1. logy webového servera,
2. dáta posielané v HTTP dopytoch,
3. aktuálne využitie systémových zdrojov.

Dôvod výberu práve týchto zdrojov je vytvorenie čo najkomplexnejšieho modelu monitorovaného systému. Pomocou dát z definovaných zdrojov chceme detegovať najčastejšie útoky podľa *OWASP* [24], a to najmä *XSS*, *CSRF*, injektovanie a *path traversal*. Spomenuté útoky sa na napadnutom systéme prejavujú rôznymi spôsobmi a výberom uvedených zdrojov dát chceme sledovať čo najvyšší počet týchto prejavov.

Z logov webového servera zaznamenávame počty jednotlivých dopytov za daný časový interval a tiež počet unikátnych IP adries, z ktorých bol dopyt zaslaný za daný časový interval. K časovým intervalom sú rovnako vypočítané pomery unikátnych IP adries k celkovému počtu dopytov. Pri zaznamenávaní údajov sa samozrejme berú do úvahy aj intervaly s nulovým počtom dopytov.

Naším cieľom je na základe uvedených typov dát detegovať pokusy o automatizované skenovanie zraniteľností útočníkom.

Ďalšou časťou je monitorovanie počtu chybových udalostí za jednotku času. Chybová udalosť je priamym ukazovateľom abnormality aktuálneho stavu.

Abstrakcia nad hodnotami v HTTP dopytoch je vytvorená tak, aby sme z bezpečnostných dôvodov zanedbávali konkrétne hodnoty zasielaných parametrov. Pre hodnoty parametrov zaznamenávame tri hodnoty reťazcov:

1. počet alfanumerických znakov,
2. počet interpunkčných znamienok,
3. počet ostatných znakov.

Z týchto hodnôt potom zaznamenávame pomery podobné pomeru spomenutého v sekcii 2.4.2. Metriky v našom návrhu sú však detailnejšie, pričom konkrétne ide o pomer počtu alfanumerickcých znakov k počtu všetkých znakov, pomer počtu interpunkčných znamienok k počtu alfanumerických znakov a pomer počtu ostatných znakov k počtu alfanumerických znakov. Hodnoty sú zaznamenávané pre každý parameter zvlášť kvôli rozličnej charakteristike vstupu rôznych parametrov.

Mená parametrov v dopytoch zaznamenávame v takej podobe v akej boli prijaté. Pomocou prijatých hodnôt tak môžeme monitorovať mieru výskytu jednotlivých parametrov a prijatie nevalidných parametrov.

Pre systémové zdroje zaznamenávame rôzne hodnoty označujúce mieru ich aktuálneho využitia. Konkrétne monitorujeme:

- CPU (aktuálne percentuálne vyťaženie),
- pamäť (aktuálne percentuálne využitie),
- sieťové rozhranie (počet prijatých/odoslaných paketov, počet prijatých/odoslaných bytov, počet chýb počas prijímania/odosielania, počet zahodených paketov, ktoré mali byť prijaté/odoslané),
- disk (počet čítaní/zápisov, počet prečítaných/zapísaných bytov, čas strávený čítaním/zápisom).

Do úložiska zaznamenávame rozdiel aktuálnych hodnôt a hodnôt z posledného merania.

Ďalšou úlohou zberačov je vytváranie antigénov, ktoré primárne slúžia pre identifikáciu konkrétnej udalosti v systéme. Ako antigén sme navrhli IP adresu HTTP dopytu, pretože ňou dokážeme do určitej miery spojiť akcie s jedným používateľom. Využitie IP adresy je bohužiaľ obmedzené pri NAT a preto je možné IP adresu doplniť. Jednou z možností je využitie hlavičky *user-agent*, či hodnoty *cookie* obsahujúcej identifikátor relácie ako napríklad pomerne často používaná *PHPSESSID*. Konkrétna hodnota je však závislá od špecifických vlastností monitorovanej PHP aplikácie.

4.3.2 Úložisko metadát

Z dôvodu modularity navrhovaného systému je potrebné špecifikovať úložisko, ktoré bude udržiavať zaznamenané dáta vo formáte akceptovanom všetkými zapojenými komponentmi.

Pre tieto účely sme sa rozhodli využiť jednoduchý koncept zdieľanej množiny vektorov. Množina je organizovaná pomocou jasne definovaných unikátnych kľúčov, ktoré identifikujú pôvod a význam jednotlivých vektorov.

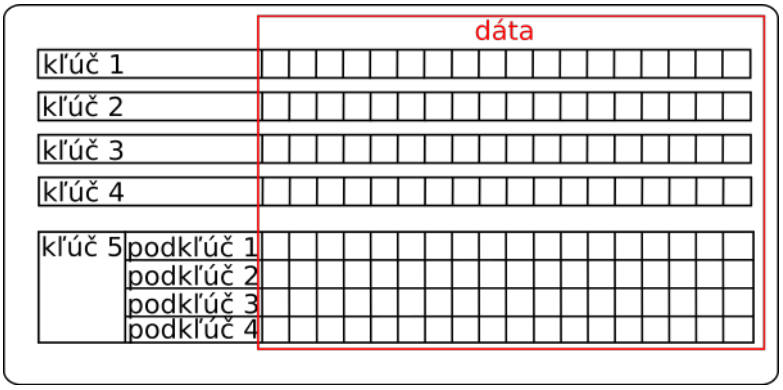
Každý vektor predstavuje front s obmedzenou dĺžkou, pričom pri prekročení maximálnej dĺžky sú staršie údaje vymazané. Existencia kľúčov je trvalá, zamieňané sú len dáta vo vektoroch.

Pre prípady, kedy je potrebná dočasná existencia vektora sme navrhli mutáciu opísaného úložiska. Trvalé kľúče v tomto prípade neukazujú na vektor, ale na množinu dočasných kľúčov. Vektory sú potom mapované pre jeden trvalý a jeden dočasný kľúč, pričom existencia vektora je podmienená existenciou dočasného kľúča. Môžu tak byť zamieňané nielen dáta vo vektoroch, ale aj dočasné kľúče.

Vizualizácia usporiadania úložiska metadát je vyobrazená na obrázku 4.3.

4.3.3 Analýza metadát

Komponent pre analýzu metadát sa skladá aspoň z jedného agentu typu *analyzátor*.



Obr. 4.3: Usporiadanie úložiska metadát.

Každý analyzátor načítava a analyzuje celý obsah všetkých vektorov z úložiska. Analýza je vykonaná pre všetky dáta rovnako bez ohľadu na ich zdroj a jej cieľom je vytvorenie vstupu pre serverovú časť vo forme signálov.

Signály v našom návrhu reprezentujeme tak ako autori algoritmu v [19]. Každý signál predstavuje metriku, ktorá je vo forme reálneho čísla normalizovaného do intervalu $< 0, 100 >$. Pre každú z metrík je potrebné vykonať mapovanie. Mapovaním rozumieme priradenie jedného z typov signálov opísaných v sekcii 3.2.5 konkrétnej metrike. Pre tento účel využívame najmä zdroj údajov, určeným kľúčom vektora v úložisku metadát.

V tejto časti návrhu sme vytvorili jeden analyzátor určený pre štatistickú analýzu zachytených metadát. Analyzátor pre jednotlivé vektory vypočítava:

- maximálnu hodnotu vektora,
- štandardnú odchýlku,
- lineárnu regresiu,
- maximálnu odchýlku od hodnoty lineárnej regresie,
- priemernú hodnotu vektora,
- medián vektora.

4.3.4 Sumarizácia metrík a ich kódové označenia

V tabuľke 4.1 sú sumarizované metriky zachytávané z navrhnutých zdrojov údajov. Spolu s popismi metrík sú v tabuľke aj kódové označenia jednotlivých

Tabuľka 4.1: Sumarizácia metrík zo všetkých navrhnutých zdrojov údajov spolu s ich kódovými označeniami.

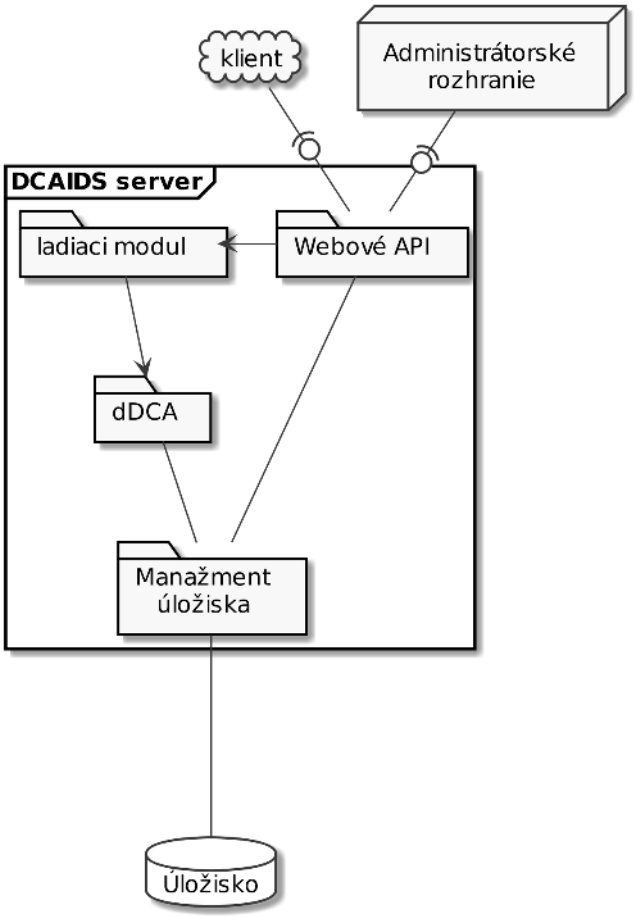
Metrika	**Kód**
Pomer počtu alfanum. znakov k počtu všetkých znakov	*alpha_ratio*
Pomer počtu špeciálnych znakov k počtu alfanum. znakov	*special_ratio*
Pomer počtu interpunk. znamienok k počtu alfanum. znakov	*interp_ratio*
Miera zmeny vyťaženia CPU za jednotku času	*cpu*
Miera zmeny zaplnenia RAM za jednotku času	*memory*
Počet odoslaných paketov po sieti za jednotku času	*packets_sent*
Počet prijatých paketov po sieti za jednotku času	*packets_rect*
Počet odoslaných bytov po sieti za jednotku času	*bytes_sent*
Počet prijatých bytov po sieti za jednotku času	*bytes_recv*
Počet chýb počas prijímania dát po sieti za jednotku času	*errin*
Počet chýb počas odosielania dát po sieti za jednotku času	*errout*
Počet zahod. paketov na odoslanie po sieti za jednotku času	*dropout*
Počet zahod. paketov na prijatie po sieti za jednotku času	*dropin*
Počet zápisov na disk za jednotku času	*write_count*
Počet čítaní z disku za jednotku času	*read_count*
Počet zapísaných bytov na disk za jednotku času	*bytes_write*
Počet prečítaných bytov z disku za jednotku času	*bytes_read*
Čas strávený zápisom na disk za jednotku času	*write_time*
Čas strávený čítaním z disku za jednotku času	*read_time*
Počet prijatých dopytov za jednotku času	*req_cnt*
Pomer počtu unik. IP a počtu dopytov za jednotku času	*ip_to_req_ratio*
Pomer počtu úsp. odp. k počtu dopytov za jednotku času	*success_status_ratio*
Počet klientských chýb za jednotku času	*client_err_cnt*

metrík, ktoré budú používané v nasledovných sekciách.

4.4 Serverová časť systému

Serverová časť systému je využitá pre spracovanie dát z klientských častí systému a následné určenie udalosti za normálnu alebo anomáliu.

Ako jadro serverovej časti využívame AIS implementovaný vo forme dDCA, u ktorého vieme deterministicky určiť priebeh jednotlivých elementov. Týmto krokom chceme odstrániť zdroje náhodnosti samotného algoritmu, a tým zjednodušiť ohodnotenie ostatných častí návrhu.



Obr. 4.4: Diagram architektúry serverovej časti navrhovaného systému.

Pre serverovú časť systému sme navrhli architektúru zobrazenú na obrázku 4.4 skladajúcu sa z nasledujúcich častí:

1. webového API,
2. komponentu pre správu úložiska,
3. komponentu dDCA,
4. administrátorského rozhrania,
5. ladiaceho modulu.

V nasledovných sekciách opíšeme jednotlivé komponenty podrobnejšie.

Súčasťou serverovej časti je tiež oddelené úložisko, ktoré je primárne určené na perzistenciu vstupu z klientskej časti a výstupu z analýzy týchto dát.

4.4.1 Webové API

Z dôvodu distribuovanej architektúry navrhovaného systému je nevyhnutné určiť spôsob prenosu dát medzi serverovou a klientskou časťou. Keďže navrhujeme systém detegujúci anomálie v reálnom čase, ako formát komunikácie sme sa rozhodli využiť webové API. Prostredníctvom API serverová časť sprístupňuje potrebnú funkcionalitu klientskej časti, keďže v našom návrhu prebieha len jednosmerná komunikácia.

Webové API je tiež využité pre komunikáciu s administrátorským rozhraním, ktoré je však z hľadiska architektonického návrhu súčasťou serverovej strany. Klientská časť využíva jednu službu webového API a to službu pre uloženie vzorky. Služby pre administrátorské rozhranie budeme definovať v ďalšej etape návrhu.

4.4.2 Komponent pre správu úložiska

Pre potreby analýzy vstupných dát sme sa rozhodli pracovať s úložiskom ako s frontom s neobmedzenou dĺžkou. Komponent pre správu úložiska má za úlohu vytvoriť rozhranie k perzistentnému úložisku a tak zaistiť nezávislosť od konkrétnej technológie využitej pri implementácii. Pri dopytoch do úložiska sú získané dáta transformované do dátových štruktúr využívaných v ďalších častiach systému.

Komponent taktiež udržiava ukazovateľ naposledy spracovávaných dát, takže pri každom dopyte dostáva systém len nové dáta a dosahujeme tak žiadanú funckionalitu frontu.

4.4.3 Komponent dDCA

Komponent dDCA je z pohľadu funckcionality najdôležitejším elementom serverovej časti navrhovaného riešenia. Jeho úlohou je z dát získaných zo servera vytvoriť výstup, pomocou ktorého bude udalosť klasifikovaná ako normálna alebo anomália. Algoritmus je spúšťaný periodicky pre všetky dáta, ktoré boli zachytené od posledného spustenia.

Priebeh algoritmu je podobný ako u pôvodného DCA opísaného v sekcii 3.2.5, pričom v tejto sekcii algoritmus opíšeme na nižšej úrovni abstrakcie.

Tabuľka 4.2: Váhovanie jednotlivých typov signálov pri vytváraní kumulovaných výstupných signálov.

Signál	**PAMP signál**	**Signál nebezpečenstva**	**Signál bezpečia**
CMS	2	1	2
polo-zrelá DC	0	0	3
zrelá DC	2	1	-3

V prvej časti je vytvorená populácia DC, pričom pre každú z buniek je vygenerovaná hranica migrácie. Po vytvorení buniek sú v cykle aplikované jednotlivé vzorky na každú z buniek, antigény a signály sú pritom spracovávané iným spôsobom.

Antigén je uložený do vektora antigénov konkrétnej bunky. Signál je spracovaný pripočítaním jeho hodnoty k premenným predstavujúcim zrelé cytokíny, polozrelé cytokíny a CSM. Pri pripočítavaní k premenným sú hodnoty signálu váhované na základe ich typu, a tiež premennej, ku ktorej sú pripočítavané. Počiatočné hodnoty pre váhy sme vybrali na základe odporúčaní autorov algoritmu [14] a hodnoty sú zhrnuté v tabuľke 4.2. Predpokladáme však zmenu týchto hodnôt v etape ladenia nastavení nášho systému.

Ak je hodnota CSM vyššia ako hranica migrácie, DC je pridaná do vektora zmigrovaných buniek. Pre každú zmigrovanú bunku je určený jeden z výsledných stavov na základe porovnania premenných predstavujúcich jednotlivé hodnoty cytokínov.

Po dokončení žiadaného počtu iterácií je pre každý z antigénov, ktoré boli načítané, vytvorená hodnota MCAV, ktorá predstavuje mieru abnormality konkrétneho antigénu a môžeme ju definovať vzťahom 4.1. Vo vzťahu symbol a predstavuje konkrétny antigén, $C_m(a)$ predstavuje počet zrelých DC, ktoré obsahujú daný antigén a $C(a)$ predstavuje počet všetkých DC, ktoré obsahujú daný antigén.

$$MCAV(a) = \frac{C_m(a)}{C(a)} \tag{4.1}$$

Hranicu MCAV, ktorá určuje kedy je antigén označený ako normálny alebo anomália, aproximujeme hodnotou 0.7 na základe testovaní autorov algoritmu [14].

4.4.4 Ladiaci modul

Oproti pôvodnému návrhu dDCA, ktoré neobsahuje žiadnu formu učenia [14] sme sa rozhodli v našom návrhu algoritmus obohatiť o určitú formu ladenia. Ako jadro ladiaceho modulu sme sa rozhodli využiť evolučný algoritmus.

Primárnym miestom optimalizácie sme určili váhovanie vstupných signálov v závislosti od ich zdroja pričom pod pojmom zdroj máme na mysli kombináciu konkrétneho zberača a metriky pred vstupom do analyzátora.

Jedinec predstavuje množinu váh pre každý zo zdrojov prítomných v systéme a môžeme ho reprezentovať binárnym reťazcom o veľkosti σ. Táto veľkosť je určená vzťahom 4.2.

$$\sigma = \lambda * \lceil \log_2(\theta) \rceil \tag{4.2}$$

Symbol λ predstavuje počet zdrojov vstupujúcich do ladenia. Pozícia zdroja vo vektore je určená abecedným poradím kódových označení metrík zo sekcie 4.3.4. Symbol θ reprezentuje počet rôznych hodnôt váh, ktoré môžu byť priradené zdroju. Spôsob prevodu konkrétneho identifikátora θ_x na hodnotu váhy zdroja je opísaný vzťahom 4.3. Vo vzťahu je $w(p)$ váha zdroja p jedinca x, θ_x je konkrétny identifikátor pre zdroj p jedinca x.

$$w(p) = \begin{cases} \frac{1}{\theta_x} + 1, & \text{ak } \theta_x \leq \frac{\theta}{2} \\ \theta_x + 1, & \text{inak} \end{cases} \tag{4.3}$$

Fitness funkcia je formalizovaná vo vzťahu 4.4, kde x predstavuje jedinca, $\alpha(x)$ predstavuje TPR pre jedinca x a $\beta(x)$ predstavuje FPR jedinca x. Hodnota c je konštanta, ktorá určuje mieru penalizácie FPR pre konkrétne prostredia nasadenia systému.

$$f(x) = \alpha(x) - c\beta(x) \tag{4.4}$$

Ako spôsob kríženia jedincov sme určili formu rulety, a to z dôvodu nízkej pravdepodobnosti uviaznutia v lokálnom optime. Na operáciu kríženia sme sa rozhodli použiť jednobodové kríženie a na operáciu mutácie zámenu náhodného bitu jedinca. Pre nahradenie sme vybrali orezanie. Prvotnú populáciu vytvárame náhodne s uniformným rozdelením. Pravdepodobnosť kríženia, mutácie

a veľkosť množiny nahradených jedincov bola určená v etape testovania ladiaceho algoritmu.

Základom fungovania algoritmu sú označené dáta, ktorých zdroj môže byť dataset, alebo ručne označené dáta administrátorom. Z tohto dôvodu je administrátorské rozhranie úzko prepojené s ladiacim modulom.

Kapitola 5

Experimentálne overenie navrhnutého systému

5.1 Implementácia prototypu návrhu

Pre implementáciu prototypu klientskej aj serverovej časti navrhovaného systému sme sa rozhodli využiť jazyk Python, ktorý sme vybrali hlavne kvôli jeho flexibilite pri vytváraní prototypov a dobrej podpore externých knižníc. Hlavným zámerom implementácie prototypu je overenie cieľov zadaných v sekcii 4.1, a preto v nej zanedbávame interaktívnu zložku systému.

5.1.1 Implementácia klientskej časti

Pre zachovanie čo najväčšej modularity implementácie sme špecifikovali rozhranie pre oba druhy agentov predstavených v sekcii 4.3. Vytvorili sme tiež prostredie, pomocou ktorého pre zahrnutie agenta do systému stačí uviesť meno jeho modulu.

Každý z agentov typu *zberač* je implementovaný ako samostatné vlákno, ktoré sleduje zmeny priradeného zdroja údajov v reálnom čase.

Pri načítavaní logu webového servera zanedbávame bežnú konvenciu rotácie logových súborov a predpokladáme, že údaje budú vkladané do rovnakého súboru.

Načítavanie dát z HTTP dopytov sme v pôvodnom návrhu chceli spojiť s monitorovaním logov webového servera nasadením jedného z nástrojov *mod_dumpio* alebo *mod_security*. Od tohto zámeru sme ustúpili, keďže má dve hlavné nevýhody: webový server by do súboru zapisoval enormné množstvo dát a taktiež by boli perzistentne zaznamenávané všetky parametre zasielané používateľom, zahrňujúc heslá. Rozhodli sme sa preto implementovať jednoduchý zachytávač paketov využitím knižnice *scapy*, pričom sme sa inšpirovali riešením z [23].

Úložisko metadát sme implementovali pomocou asociatívneho poľa, ktoré udržiavame len v pamäti. Pre účely prototypu nie je potrebné využiť perzistentné úložisko, keďže zachytávame len veľmi obmedzený časový úsek a nemusíme riešiť špeciálne prípady, napr. hardvérové poruchy.

Agenty zo skupiny *analyzátory* sú navrhnuté pomocou návrhového vzoru *observer*, kde úložisko metadát spúšťa analýzu na registrovaných analyzátoroch pri splnení daných podmienok. V našom prípade je to obmenenie daného počtu prvkov vo fronte úložiska. Po vytvorení signálov sú tieto transformované do JSON formátu a pomocou HTTP POST dopytu poslané na serverovú časť.

5.1.2 Implementácia serverovej časti

Serverová časť systému je implementovaná ako webová aplikácia. Pre implementáciu sme vybrali mikrorámec *Flask*. Rámec sme vybrali hlavne kvôli nízkym nárokom na zdroje, nakoľko v našom prípade nepotrebujeme pokročilé vlastnosti robustnejších rámcov.

Ako perzistentné úložisko sme použili *NoSQL* databázu *mongo*, pričom sme však v implementácii minimalizovali závislosť na tejto konkrétnej databáze. Databázu sme vybrali hlavne kvôli priamej podpore JSON formátu, ktorý je využitý pre komunikáciu a taktiež flexibilite pri vytváraní prototypu. Softvérový návrh však dovoľuje využitie akejkoľvek databázy minimálnou úpravou kódu.

Ako základ implementácie algoritmu dDCA sme využili príklad implementácie v [4]. Od príkladu sme sa však po krátkom čase odklonili, nakoľko podľa nášho názoru neimplementuje špecifikáciu vhodne. Najväčším nedostatkom je pre nás oddelenie trénovacej a testovacej časti, čo je v rozpore s autorským opisom algoritmu.

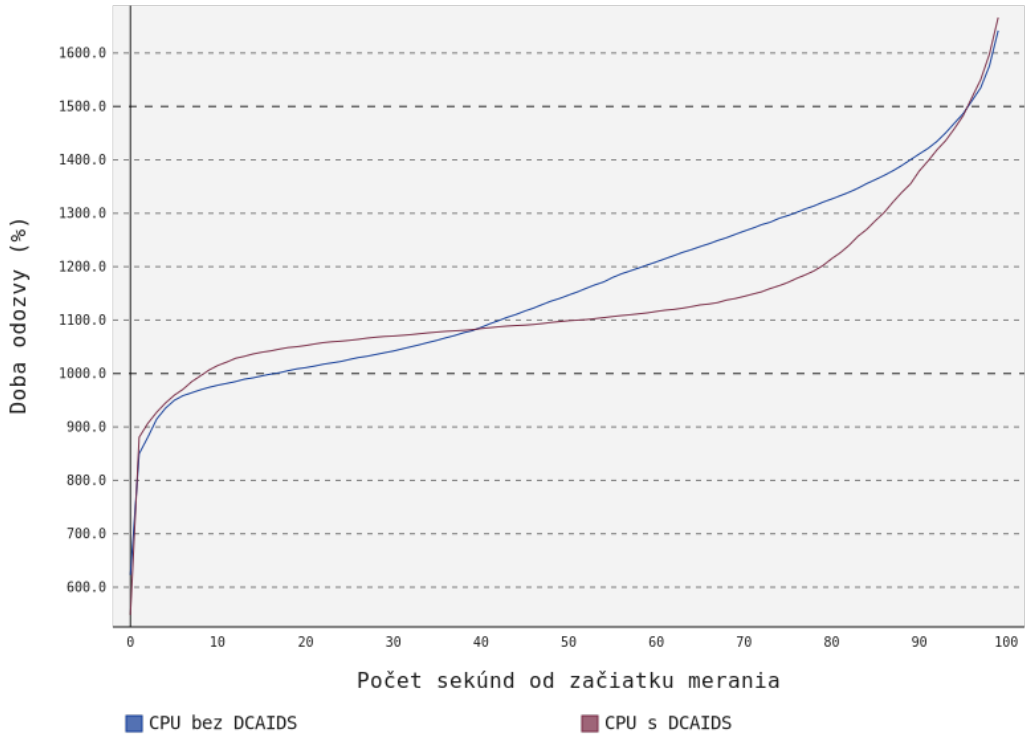
5.2 Overenie dopadu na monitorovaný systém

Jeden z definovaných cieľov navrhovaného riešenia je minimalizácia zaťaženia monitorovaného systému. Z tohto dôvodu sme označili overenie dopadu klientskej časti na výkon monitorovaného systému ako prioritnú úlohu po dokončení implementácie.

Tabuľka 5.1: Parametre virtuálneho stroja využitého na overenie dopadu implementácie klientskej časti navrhovaného systému na monitorovaný systém.

CPU	Intel(R) Core(TM) i7-4600U CPU @ 2.10GHz
Počet jadier CPU	1
RAM	1.5GB
OS	CentOS Linux release 7.1.1503 (core)
Virtualizačné prostredie	VirtualBox 5.0.10_OSEr104061

Za sledované metriky testu sme určili využitie CPU a dobu odozvy servera na HTTP dopyt, pričom náš cieľ je sledovať tieto metriky počas záťaže na monitorovaný systém - webový server. Webový server sme nasadili na virtuálny stroj, ktorého parametre sú zobrazené v tabuľke 5.1. Na vytvorený virtuálny stroj sme nasadili PHP aplikáciu *Wordpress*, ktorá je v súčasnej dobe na webových systémoch veľmi často nasadzovaná, podľa [13] až v 25% celého webu.



Obr. 5.1: Doba odozvy HTTP dopytu počas výkonnostného testu.

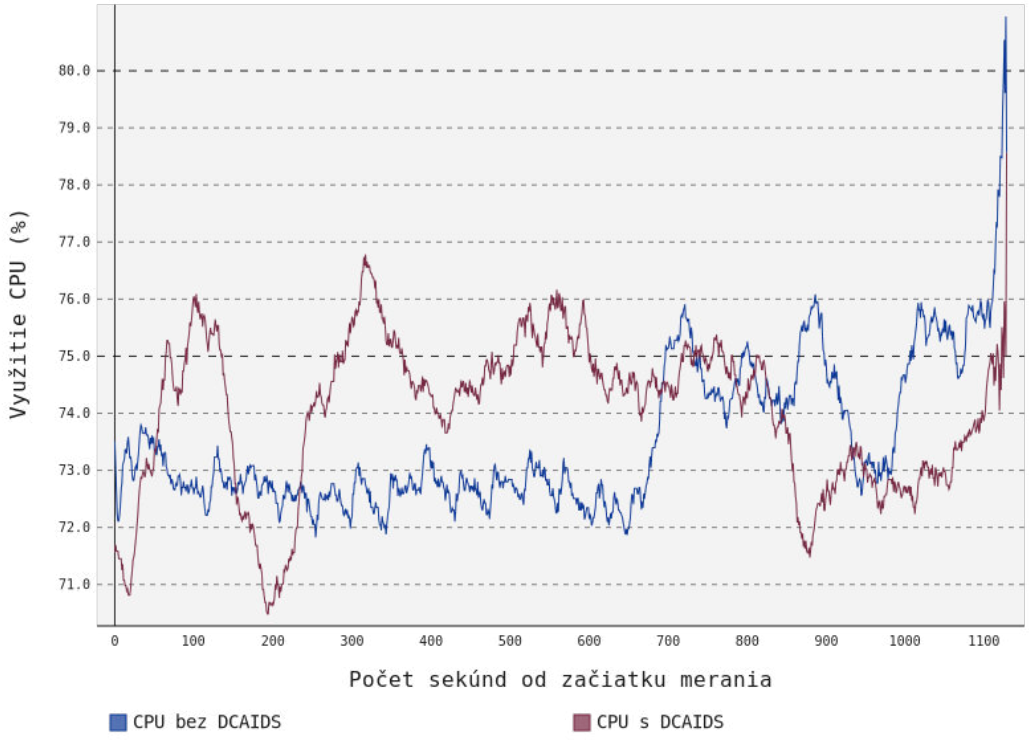
Test sme vykonali nástrojom *ab*, pomocou ktorého sme vygenerovali 10 000 dopytov v desiatich súbežných reláciách, pričom sme ten istý scenár zopakovali dvakrát: bez nasadenia implementovanej aplikácie a s nasadením a spustením implementovanej aplikácie.

Tabuľka 5.2: Priemerné hodnoty metrík overenia dopadu implementácie klientskej časti navrhovaného systému na monitorovaný systém.

	Využitie CPU (%)	**Doba odozvy (ms)**
Bez nasadeného IDS	73.9560	1135.1645
S nasadeným IDS	73.4491	1165.2988

Pomocou nástroja *ab* sme tiež zaznamenali dobu odozvy webového servera na monitorovanom virtuálnom stroji. Výsledky sú zobrazené na obrázku 5.1 a sú zoradené podľa času odozvy.

Vyťaženie CPU monitorovaného virtuálneho stroja sme zaznamenali nástrojom *dstat*, pričom výsledky sú zobrazené na obrázku 5.2.



Obr. 5.2: Vývoj zaťaženia CPU monitorovaného systému v čase počas výkonnostného testu.

V tabuľke 5.2 môžeme vidieť priemerné hodnoty zaznamenávaných metrík. Z výsledkov testu môžeme zhodnotiť, že implementácia klientskej časti má na monitorovaný systém z hľadiska využitia CPU a doby odozvy na HTTP dopyty zanedbateľný dopad.

5.3 Opis metódy testovania pomocou datasetov

Pre ohodnotenie niektorých častí návrhu sme sa rozhodli využiť tri dostupné datasety. Prvým je dataset *CSIC 2010* [25] (ďalej len *CSIC*), ktorý je zostavený z dopytov na internú e-comerce aplikáciu autorov. Anomálne dopyty v datasete zachytávajú veľké množstvo typov útokov ako injektovanie, *XSS*, *CSRF*, *buffer overflow* a mnohé ďalšie. V datasete je možné spojiť dopyty jednotlivých relácií pomocou parametra *JSESSIONID*. Dataset síce obsahuje URL na ktorú boli dopyty smerované, nedisponujeme však špecifikáciou aplikácie autorov, a tak nevieme určiť, ktoré URL sú validné a ktoré nie. Dataset tiež neobsahuje čas odoslania dopytu. Situáciu preto už nie je možné zreplikovať exaktne.

Druhým datasetom je *HttpParamsDataset* [27] (ďalej len *Smihla*). Dataset je zameraný na útoky vykonané poslaním anomálneho reťazca v dopyte. Obsahuje útoky ako *XSS*, *CSRF*, *path traversal* a podobne. Každý útok je obsiahnutý v jedinom dopyte a nie je možné identifikovať reláciu. Dataset tiež neobsahuje časy odoslania jednotlivých dopytov a neobsahuje ani meno parametra, v ktorom boli dáta odoslané.

Dva opísané datasety sme využili dvomi spôsobmi: statickou analýzou vybranej časti datasetu a preposlaním dopytov na náš testovací webový server nasadený na virtuálnom stroji opísanom v sekcii 5.2. Pre odosielanie na testovací webový server sme implementovali testovací skript, ktorý preposiela dopyty z datasetu v náhodných časových intervaloch, v priemere sto dopytov za sekundu. Dáta posiela pomocou metódy *POST*. Pre dataset *Smihla* sme dopyt zostrojili pridaním dát do parametra *login*.

Keďže sú oba datasety anotované, označenie dopytu sme posielali v zvláštnom parametri pre účely ohodnotenia výstupu a ladenia.

Tretím datasetom je *StarWarsKidDataDump* [3] (ďalej len *SWK*). Tento dataset je pomerne obsiahlym log súborom webového servera *Apache* a využili sme ho pre overenie metrík získaných z logov webového servera. Dataset nie je anotovaný a tiež nemáme informáciu či a aké útoky obsahuje. Môžeme však predpokladať, že dataset je z veľkej časti normálny.

5.4 Overenie metrík klientskej časti systému

Nutnou podmienkou korektného fungovania navrhnutej aplikácie je správne mapovanie navrhnutých metrík na jednotlivé typy signálov. Pre čo najlepšie splnenie tejto úlohy sme vykonali testy jednotlivých metrík, pričom sme sa zamerali na rozdiel hodnôt jednotlivých metrík pri normálnej prevádzke a počas útoku. Ďalším nezanedbateľným cieľom je redukcia navrhnutej množiny metrík.

5.4.1 Overenie metrík využitia systémových zdrojov

Keďže metriky využitia systémových zdrojov majú zachytávať automatizované skenovacie skripty a útoky hrubou silou, pre tento test sme nepoužili žiadny z opísaných datasetov. Využili sme nástroj *WPScan*, pomocou ktorého sme vykonali slovníkový útok spustený v päťdesiatich vláknach. Ako testovacie prostredie sme použili virtuálny stroj opísaný v sekcii 5.2, pričom požiadavky boli smerované na aplikáciu *Wordpress*. Tri minúty sme zachytávali všetky definované metriky počas normálnej prevádzky a následne tri minúty počas opísaného útoku. V tabuľke 5.3 sú zobrazené namerané výsledky priemerných hodnôt metrík počas normálnej prevádzky a v tabuľke 5.4 hodnôt počas anomálnej prevádzky.

Výsledky ukazujú, že takmer všetky metriky zmenili počas útoku hodnotu. Takmer všetky metriky tiež dosahovali počas útoku vyššie hodnoty, výnimkou bol medián. Test ukázal, že minimálna hodnota vektora je použiteľná len v obmedzenom rozsahu. V ďalších testoch sme preto zlúčili minimum a maximum vektora do jednej hodnoty - rozpätia hodnôt vektora.

Spozorovali sme tiež nestabilné správanie počtu prečítaných a zapísaných bytov na disk, a prijatých a poslaných bytov po sieti. Táto nestabilita sa prejavovala aj počas normálne prevádzky, čo môžeme vyčítať napríklad z vysokej hodnoty lineárnej regresie.

Z nameraných dát vyvodzujeme nevhodnosť metrík využívajúcich počty bytov pre ďalšie použitie. V ďalších testoch sa tiež zameriame na hodnoty mediánu analyzovaných metrík.

Tabuľka 5.3: Percentuálny rast metrík využitia systémových zdrojov počas normálneho používania a útoku.

Kódové označ. metriky	Lin. reg.	Štd. odch.	Min.	Max.	Priemer	Medián
cpu	0.317	2.494	-6.55	6.726	0.057	0.045
memory	0.014	0.029	-0.055	0.156	0.007	0.001
write_time	6.117	18.94	0	137.63	5.898	0.145
read_time	1.448	4.819	0	27.126	0.740	0.050
bytes_write	71246	262528	0	1352431	74641	233
bytes_read	1794	19450	0	168506	3558	51.848
write_count	4.370	10.573	0	45.367	3.854	0.066
read_count	0.213	0.858	0	4.707	0.012	0.012
bytes_sent	10861	38960	0	200800	14975	59
bytes_recv	466.89	1257	0	5926	621	57
packets_sent	4.313	12.286	0	71.569	5.164	0.838
packets_recv	7.148	22.663	0	129.607	8.632	0.870

Tabuľka 5.4: Percentuálny rast metrík využitia systémových zdrojov počas normálneho používania a útoku.

Kódové označ. metriky	Lin. reg.	Štd. odch.	Min.	Max.	Priemer	Medián
cpu	2.363	16.807	-52.49	60.377	1.424	0.032
memory	1.247	1.945	-1.814	7.450	0.752	0.133
write_time	823148	2.86e+6	0	1.62e+7	754799	0
read_time	6155.53	12676	0	59558	5550.32	43.434
bytes_write	7.51e+6	2.18e+7	0	1.04e+8	6.79e+6	0
bytes_read	1.55e+7	3.07e+7	0	1.47e+8	1.39e+7	120979
write_count	939.35	3339.82	0	16721	843.601	0
read_count	692.19	1990.67	0	14807	667.045	11.225
bytes_sent	35836	84822	0	261836	51872	5445
bytes_recv	4061.6	10109	0	30126	5506	184
packets_sent	25.365	53.497	0	173.549	32.477	2.787
packets_recv	30.481	62.522	0	200.862	39.725	3.699

Tabuľka 5.5: Priemerné hodnoty metrík dát HTTP dopytov počas normálnej prevádzky.

Kódové označ. metriky	Lineárna regresia	Štd. odchýlka	Rozpätie hodnôt	Priemer	Medián
alphanum_ratio	2.970e-6	0.03514	0.1444	0.9582	0.9730
interp_ratio	2.421e-6	0.01088	0.0730	0.0098	0.0077
special_ratio	-4.274e-6	0.04000	0.1861	0.0413	0.0236

Tabuľka 5.6: Priemerné hodnoty metrík dát HTTP dopytov počas anomálnej prevádzky.

Kódové označ. metriky	Lineárna regresia	Štd. odchýlka	Rozpätie hodnôt	Priemer	Medián
alphanum_ratio	3.02e-05	0.1159	0.8731	0.92651	0.9694
interp_ratio	-2.61e+14	7.00e+17	7.33e+18	9.23e+16	0.0094
special_ratio	-2.60e+14	7.00e+17	7.32e+18	9.24e+16	0.0265

5.4.2 Overenie metrík dát HTTP dopytov

V ďalšom kroku sme overili metriky dát HTTP dopytov. Overenie sme vykonali pomocou datasetu *CSIC*. Postupne sme načítali prvých 10 000 záznamov z normálneho a anomálneho datasetu a vyhodnotili sme priemerné hodnoty metrík pre každý z datasetov. V tabuľke 5.5 sú zobrazené priemerné hodnoty metrík pre normálny dataset a v tabuľke 5.6 pre anomálny.

Pozorujeme prudký nárast priemerných hodnôt metrík *interp_ratio* a *special_ratio*. Môžeme konštatovať, že metriky vhodne odzrkadľujú anomálne správanie monitorovaného systému. Pre metriku *alphanum_ratio* pozorujeme komplexnejší vývoj hodnôt. Priemerná hodnota a hodnota lineárnej regresie klesla, ostatné metriky však stúpli.

Zmena hodnôt mediánu metrík pre normálny a anomálny dataset je minimálna. Z tohto dôvodu, a taktiež nedeterninistických výsledkov v teste v sekcii 5.4.1, sme sa rozhodli hodnotu mediánu z návrhu odstrániť.

5.4.3 Overenie metrík logov webového servera

Pre overenie hodnôt metrík logov webového servera sme využili dataset *SWK*, v ktorom však prevádzka nie je úplne normálna. Bohužiaľ sme neboli schopní získať dostatočne veľký dataset, ktorý by bol zaručene normálny. Napriek tomu

Tabuľka 5.7: Priemerné hodnoty metrík logov webového servera počas sieťovej prevádzky.

Kódové označ. metriky	Lineárna regresia	Štd. odchýlka	Rozpätie hodnôt	Priemer
req_cnt	5.18392e-5	0.9054	8.7412	0.9173
ip_to_req_ratio	-3.5567e-7	0.0494	0.5622	0.9936
succss_status_ratio	-1.2984e-6	0.1128	0.6632	0.9711
client_err_cnt	3.93111e-6	0.2253	2.1370	0.0400

je pre nás tento dataset cenný a očakávame získanie hodnôt metrík najmä pre účely škálovania. Hodnoty síce nebudú presné, budú sa však blížiť k hodnotám normálneho datasetu. Pri škálovaní preto s výsledkami pracujeme so zohľadnením charakteru datasetu.

Pri teste sme využili prvých 10 000 záznamov datasetu a v tabuľke 5.7 sú zobrazené priemerné hodnoty zaznamenané pri teste.

Jednotlivé hodnoty majú povahu ako v predchádzajúcich testoch, a preto môžeme konštatovať, že metriky vhodne reprezentujú zväčša normálny dataset.

5.4.4 Vývoj hodnôt metrík na základe parametrov úložiska metadát

Nevyhnutnou časťou ďalšieho testovania je určenie parametrov úložiska metadát, keďže sa od nich odvíjajú hodnoty všetkých analyzovaných metrík. Pre otestovanie týchto parametrov sme využili metriky dát HTTP dopytov pomocou datasetu *CSIC*, pričom sme simulovali dopyty pomocou zasielania dát do systému v dvoch častiach. V prvej časti sme použili len normálne dopyty a v druhej celý dataset, aj s anomálnymi dopytmi.

Otestovali sme kombináciu štyroch veľkostí úložiska (400, 600, 800, 1 000) a troch frekvencií analýzy ($^1/_5$, $^1/_{10}$, $^1/_{20}$), pričom pod pojmom frekvencia analýzy máme na mysli takú časť vektora, ktorá je vymenená medzi jednotlivými analýzami. V tabuľke 5.8 sú zobrazené výsledky testu zoradené podľa štandardnej odchýlky hodnôt pri teste s normálnym datasetom.

Pôvodne sme pre určenie parametrov chceli využiť aj mieru rastu metrík pre celý dataset oproti normálnemu datasetu. Rast bol však vysoký aj

Tabuľka 5.8: Výsledky testu parametrov úložiska metadát použitím metrík dát HTTP dopytov.

Veľkosť vektora	Frekvencia analýzy	Priemer hodnôt normálneho datasetu	Štd. odchýlka normálneho datasetu	Priemer hodnôt anomálneho datasetu
400	$^1/_{20}$	0.0393807	0.0966003	9.8631e+17
400	$^1/_{10}$	0.0394212	0.0967531	9.87451e+17
400	$^1/_5$	0.039461	0.0967941	9.90192e+17
600	$^1/_{20}$	0.0410106	0.10269	1.09961e+18
600	$^1/_{10}$	0.0410359	0.102806	1.10163e+18
600	$^1/_5$	0.0412468	0.103346	1.10681e+18
800	$^1/_{20}$	0.0421395	0.107534	1.16427e+18
800	$^1/_{10}$	0.0422074	0.107645	1.17068e+18
800	$^1/_5$	0.0424263	0.108336	1.18194e+18
1000	$^1/_{20}$	0.0430198	0.111298	1.216e+18
1000	$^1/_{10}$	0.0431532	0.111694	1.21763e+18
1000	$^1/_5$	0.0433343	0.112413	1.22068e+18

v najhoršom prípade, preto sme sa zamerali na štandardnú odchýlku. Z pohľadu priemerných hodnôt a štandardnej odchýlky pre normálny dataset sme dosiahli najlepšie výsledky pre vektor veľkosti 400 s frekvenciou analýzy $^1/_{20}$.

Pre dané konfigurácie vyslovujeme hypotézu, že pri menšej veľkosti vektora a vyššej perióde analýzy budeme schopní znížiť počet *FP*, keďže anomálne hodnoty budú vo vektore nahradené v kratšom čase.

5.5 Určenie systémových nastavení

Ako nástroj pre určenie systémových nastavení sme využili časť normálneho a anomálneho trénovacieho datasetu *CSIC*. Pri ladení parametrov sme sa snažili o čo najmenší počet TP pri normálnom datasete a čo najväčší počet TP pri anomálnom datasete. Kvôli veľkému množstvu parametrov sme hodnoty nehľadali automatizovane, ale postupne sme ohodnocovali rôzne kombinácie manuálnymi testami. Nové kombinácie sme vyberali na základe výsledku predchádzajúceho testu.

5.5.1 Mapovanie signálov a škálovanie metrík

Na základe rozdielov medzi hodnotami nameranými v normálnom a anomálnom datasete v sekcii 5.4 sme vykonali mapovanie signálov do jednej z troch kategórií opísanych v sekcii 3.2.5. V ďalšom kroku sme na základe hodnôt prezentovaných v sekcii 5.4 vykonali škálovanie metrík. Cieľom bolo dosiahnuť hodnoty z intervalu $< 1, 100 >$. Ak škálovaná hodnota signálu predsa dosiahne hodnotu väčšiu ako 100, bude upravená na hodnotu 100.

V tabuľke 5.9 je zobrazené mapovanie jednotlivých metrík na signály spolu s hodnotou škálovania pre každý zo signálov. Pre označenie jednotlivých metrík sme použili kódové označenia zo sekcií 5.4.1, 5.4.2 a 5.4.3.

V pôvodnom návrhu sme počítali s použitím mapovania metriky pre všetky analyzované hodnoty. Pri testovaní sa však tento spôsob ukázal ako nevhodný a vyhodnotili sme, že zvýšená hodnota metrík lineárnej regresie, rozpätia hodnôt v poli a štandardnej ochýlky je vždy ukazovateľom anomálie. Preto sme tieto označili za signál nebezpečenstva, pričom mapovanie z tabuľky 5.9 sa vzťahuje len na priemernú hodnotu vektora.

Toto usporiadanie sa však tiež ukázalo ako neoptimálne, keďže sme zaznamenali veľké množstvo FP. Zmenili sme preto označenie metriky rozpätia hodnôt na signál bezpečia, pričom sme upravili hodnotu metriky umocnením na -1. Výsledky po tejto zmene boli najlepšie zo všetkých testovaných.

5.5.2 Určenie parametrov algoritmu dDCA

Pre algoritmus dDCA bolo potrebné určiť hodnoty váh pre vytváranie kumulovaných výstupných signálov, hodnotu veľkosti populácie dendritických buniek a rozsah z ktorého je vybraná migračná hodnota jednotlivých dendritických buniek.

Ako iniciálne hodnoty testovania váh pre vytvárania kumulovaných výstupných signálov sme vybrali hodnoty publikované autormi algoritmu v [14]. Tieto hodnoty sú zhrnuté v tabuľke 4.2.

Pre náš model sa však hodnoty ukázali ako nevhodné a manuálne sme testovali veľké množstvo rôznych kombinácií parametrov. Z testovaných hodnôt

Tabuľka 5.9: Mapovanie metrík z rôznych zdrojov na jednotlivé typy signálov s hodnotou pre škálovanie metrík.

Zdroj údajov	**Typ signálu**	**Metrika**	**Škálovanie**
HTTP dáta	PAMP signál	special_ratio	100.0
	Signál nebezp.	interp_ratio	100.0
	Signál bezp.	alphanum_ratio	100.0
Logy webového servera	PAMP signál	-	-
	Signál nebezp.	req_cnt	0.1
		client_err_cnt	1.0
	Signál bezp.	ip_to_req_ratio	100.0
		success_status_ratio	100.0
Využitie systémových zdrojov	PAMP signál	errin	1.0
		errout	1.0
		dropin	1.0
		dropout	1.0
	Signál nebezp.	cpu	1.0
		memory	1.0
		read_count	1.0
		write_count	1.0
		read_time	1.0
		write	1.0
		packets_sent	0.1
		packets_recv	0.1
	Signál bezp.	-	-

Tabuľka 5.10: Váhovanie jednotlivých typov signálov pri vytváraní kumulovaných výstupných signálov.

Signál	**PAMP signál**	**Signál nebezpečenstva**	**Signál bezpečia**
CMS	3	1	7
polo-zrelá DC	0	0	3
zrelá DC	9	5	-3

sme najlepšie vyhodnotili kombináciu opísanú v tabuľke 5.10. V tejto oblasti cítime potrebu ďalšieho ladenia, napríklad formou strojového učenia.

Po testovaní viacerých hodnôt veľkosti populácie DC a rozsahu migračnej hranice sme dospeli veľkosti populácie 1 000, minimálnej hodnote migračnej hranice 600 a maximálnej hodnote migračnej hranice 800.

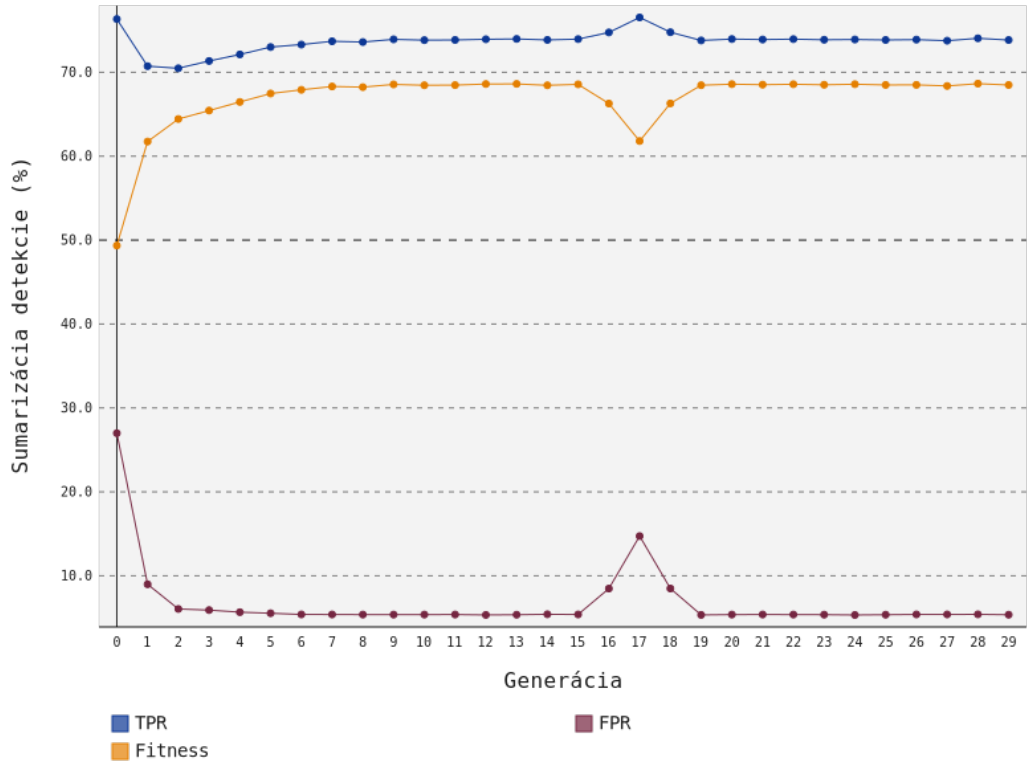
5.6 Implementácia a overenie ladiaceho evolučného algoritmu

Po určení hodnôt potrebných pre fundamentálne fungovanie riešenia sme pristúpili k implementácii evolučného algoritmu navrhnutého v sekcii 4.4.4. Pre overenie implementovaného algoritmu sme určili hodnotu pravdepodobnosti mutácie na 20% a mieru nahradenia na 80%. Tieto hodnoty sme vybrali najmä kvôli zamedzeniu možnosti uviaznutia v lokálnom optime. Veľkosť populácie sme určili na tridsať jedincov a počet generácií taktiež na tridsať. Pre parameter c zo vzťahu 4.4 sme určili hodnotu 1.

Určené parametre sme otestovali pomocou plného datasetu *CSIC*, kde sme sledovali hodnoty priemernej *TPR* a *FPR* v jednotlivých generáciách. Tieto hodnoty sú vizualizované na obrázku 5.3.

Na grafe môžeme pozorovať, že hodnoty začínajú pomerne rýchlo konvergovať k výslednej hodnote. Keďže algoritmus bol pri určených parametroch príliš náročný na čas a výpočtové prostriedky, rozhodli sme sa otestovať konfiguráciu s desiatimi jedincami a desiatimi generáciami. Hodnoty sú zobrazené na obrázku 5.4.

V tabuľke 5.11 sú zobrazené hodnoty najlepšieho jedinca poslednej generácie oboch testov. Z týchto dát môžeme usúdiť, že rozdiely sú zanedbateľné

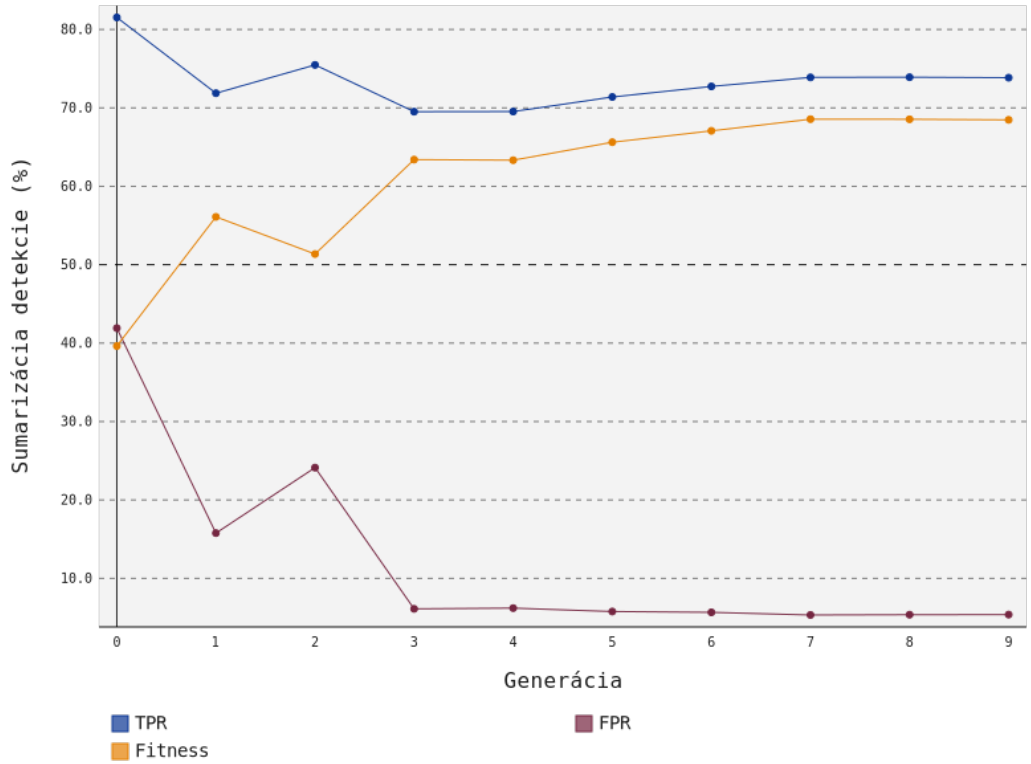


Obr. 5.3: Vývoj priemernej TPR, FPR a fitness *generácie pri teste evolučného algoritmu s tridsiatimi jedincami a tridsiatimi generáciami pre vybraný prípad ladenia.*

a menší počet jedincov a generácií za cenu zníženia času výpočtu je pre naše účely dostačujúci.

Tabuľka 5.11: Výsledky testu dvoch konfigurácií implementovaného evolučného algoritmu pre vybrané prípady ladenia.

Počet jedincov	30	10
Počet generácií	30	10
TPR najlepšieho jedinca	74.7077	74.8007
FPR najlepšieho jedinca	5.2388	5.4085
Fitness najlepšieho jedinca	69.4688	69.3921



Obr. 5.4: Vývoj prmeiernej TPR, FPR a fitness *generácie pri teste evolučného algoritmu s desiatimi jedincami a desiatimi generáciami pre vybraný prípad ladenia.*

5.7 Ladenie výsledkov detekcie pomocou parametrov úložiska metadát

Po vykonaní testu v sekcii 5.4.4 sme chceli overiť hypotézu pre hraničné kombinácie definovaných parametrov úložiska metadát z tabuľky 5.8. Test sme vykonali na časti plného datasetu *CSIC* a časti plného datasetu *Smihla* pričom sme porovnali výsledky dvoch konfigurácií datasetu. Porovnali sme tiež hodnoty pred a po ladení evolučným algoritmom. Výsledky sú zobrazené v tabuľke 5.12.

Zo získaných dát konštatujeme, že hypotéza sa ukázala ako pravdivá a zväčšením veľkosti vektora síce získame zvýšenú TPR, markantne sa však zvýši aj FPR. Pre ďalšie použitie sme preto vyhodnotili konfiguráciu s veľkosťou 400 a frekvenciou analýzy $^1/_5$ ako vhodnejšiu.

Tabuľka 5.12: Výsledky testu parametrov úložiska metadát pre dataset CSIC *a* Smihla.

Veľkosť	400		1000	
Frekvencia analýzy	$^1/_{20}$		$^1/_5$	
Ladenie	✗	✓	✗	✓
TPR (*CSIC*)	92.8737	97.1911	99.6506	99.6514
FPR (*CSIC*)	13.0106	12.8664	24.4755	21.7316
TPR (*Smihla*)	99.1236	99.2758	99.9914	100.0000
FPR (*Smihla*)	51.4585	36.4908	90.1860	69.4506

5.8 Porovnanie finálnych výsledkov so systémom PHPIDS

Ako systém, ktorého výsledky chceme porovnávať s výsledkami nášho systému sme vybrali PHPIDS. Vybrali sme ho najmä kvôli tomu, že podobne ako náš systém sa zameriava na detekciu útokov na aplikačnej úrovni, pričom je pomerne využívaný v praxi.

Pre finálny test sme využili jednoduchý *PHP* skript na ktorý sme zasielali požiadavky z datasetov. Na základe testov v predchádzajúcich sekciách sme pre systém definovali ďalšiu konfiguráciu váhovania jednotlivých typov signálov pri vytváraní kumulovaných vstupných signálov. Hodnoty konfigurácií sa nachádzajú v tabuľke 5.13. Riešenie sme overili pre obe konfigurácie.

Test prebiehal v dvoch etapách. V prvej sme poslali na testovaciu aplikáciu trénovací dataset *Smihla*. Po tomto kroku sme vykonali ladenie pomocou evolučného algoritmu a pomocou získanej konfigurácie sme vykonali detekciu na testovacom datasete *Smihla*. Aplikáciu *PHPIDS* sme overili len pomocou testovacieho datasetu *Smihla*, keďže aplikácia neobsahuje ladiaci mechanizmus. Výsledky prvej etapy sú zobrazené v tabuľke 5.14, pričom náš systém je označený ako DCAIDS.

V druhej etape sme na testovací skript poslali plný dataset *CSIC*. Pôvodne sme chceli test opäť rozdeliť na testovaciu a trénovaciu fázu, pre *CSIC* však neexistuje vhodný trénovací dataset. Výsledky druhej etapy sú zobrazené v tabuľke 5.15, pričom náš systém je opäť označený ako DCAIDS.

Zo získaných dát je viditeľné, že výstup oboch systémov sa poznateľne líši pre oba datasety V prípade nášho systému sa výstup líši aj na základe použitej

Tabuľka 5.13: Váhovanie jednotlivých typov signálov pri vytváraní kumulovaných výstupných signálov.

Signál	**Konfigurácia**	**PAMP signál**	**Signál nebezpečenstva**	**Signál bezpečia**
CMS	1	3	1	7
polo-zrelá DC		0	0	3
zrelá DC		9	5	-3
CMS	2	3	1	7
polo-zrelá DC		0	0	3
zrelá DC		9	3	-4

Tabuľka 5.14: Výsledky finálneho testu systému s porovnaním s PHPIDS pre dataset Smihla.

Systém	DCAIDS				PHPIDS
Konfigurácia	1		2		-
Ladenie	✗	✓	✗	✓	-
TPR	98.9674	98.2388	95.2130	96.7488	97.4751
FPR	52.8751	38.9312	8.0179	7.6819	0.3108

Tabuľka 5.15: Výsledky finálneho testu systému s porovnaním s PHPIDS pre dataset CSIC.

Systém	DCAIDS				PHPIDS
Konfigurácia	1		2		-
Ladenie	✗	✓	✗	✓	-
TPR	48.9990	64.9189	81.9157	87.8811	15.6034
FPR	6.9291	6.2723	14.4035	12.7115	0.5555

konfigurácie. Pri datasete *Smihla* sme dosiahli porovnateľnú TPR ako systém *PHPIDS*. Pri konfigurácii 1 bola táto hodnota pre náš systém mierne vyššia, pri konfigurácii 2 mierne nižšia.

Z pohľadu FPR bol náš systém poznateľne horší, pričom ako vhodnejšia sa jasne javí konfigurácia 2. Výsledok detekcie pre dataset *Smihla* pripisujeme povahe datasetu, v ktorom nie je možné priradiť jednotlivé útočné dopyty ku konkrétnej relácii. Útočné dopyty v datasete sú tiež výhradne textového charakteru a sú dobre zachytiteľné pomocou príznakov, na ktoré je systém *PHPIDS* špecializovaný.

Výhoda nášho systému sa ukazuje pri datasete *CSIC* kde sme dosiahli výrazne vyššiu TPR ako systém *PHPIDS*. Pri konfigurácii 2 sme dosiahli vyššiu hodnotu TPR za cenu vyššej FPR . Z výsledkov nie je zjavné, ktorá z konfigurácií je vhodnejšia. Usudzujeme, že výber konfigurácie by v tomto prípade mal záležať na konkrétnych potrebách prostredia, v ktorom je systém nasadený. Náš systém síce dosiahol vyššiu FPR ako systém *PHPIDS*, táto hodnota je však podľa nášho názoru jasne kompenzovaná obrovským zvýšením TPR. Výsledok je pravdepodobne spôsobený lepšou identifikáciou dopytov jednotlivých relácií v datasete a taktiež pomerne veľkým množstvom anomálnych dopytov, ktoré sa bez kontextu nejavia ako anomálne. Tieto dopyty sú potom pre IDS s detekčným mechanizmom na základe príznakov veľmi ťažko detegovateľné.

Ladenie pomocou *EA* zlepšilo dosiahnutý výsledok vo všetkých prípadoch. Okrem jedného prípadu môžeme konštatovať, že toto zlepšenie bolo výrazné a že ladiaci modul pomerne veľkou mierou prispel k výsledkom systému.

Na základe analýzy vlastností porovnávaných systémov a dát nazbieraných pri využití datasetov *CSIC* a *Smihla* sme vyhodnotili systém *DCAIDS* ako univerzálnejší, keďže dokázal identifikovať značnú časť útokov pre oba testované datasety pri udržaní pomerne nízkej FPR. Pri datasete *Smihla* náš systém mierne zaostával kvôli zvýšenej FPR, jasne však dominoval pri datasete *CSIC*.

Kapitola 6

Zhodnotenie

V našej práci sme predstavili druhú časť riešenia systému pre zvýšenie bezpečnosti webového systému využitím biológiou inšpirovaných metód.

V našej práci sme analyzovali problematiku potrebnú pre vytvorenie kompetentného návrhu. V kapitole 2 sme analyzovali niektoré z najznámejších a najčastejších útokov na webové systémy, prestavili sme systémy na detekciu prienikov a ich základné rozdelenie. Na záver sme analyzovali existujúce systémy na detekciu prienikov.

V kapitole 3 sme analyzovali vybrané prístupy výpočtovej inteligencie. Konkrétne sme analyzovali umelé imunitné systémy a evolučné algoritmy. Kombináciu analyzovaných prístupov sme sa rozhodli využiť v našom vlastnom návrhu.

V kapitole 4 sme podrobne predstavili návrh nášho riešenia. Ako sme uviedli vyššie, v návrhu sme sa rozhodli využiť umelý imunitný systém v kombinácii s evolučným algoritmom slúžiacim na ladenie. Konkrétne sme vybrali deterministický algoritmus dendritických buniek a v návrhu sme vytvorili model pre tento algoritmus. Navrhnutý model je určený pre webové systémy.

Na klientskej časti nášho systému sú umiestnené agenty pre zber a analýzu dát a na centrálnej časti je umiestnený samotný algoritmus dDCA určený pre detekciu anomálií. Definovali sme ladenie riešenia pomocou evolučného algoritmu, ktorý určuje váhu jednotlivých zdrojov údajov.

Pre navrhnutý systém sme vytvorili implementáciu, ktorú sme opísali v kapitole 5. Pomocou implementovaného prototypu sme overili splnenie nastolených cieľov. Vykonali test zaťaženia monitorovaného systému klientskou časťou nášho systému. Overili sme vhodnosť jednotlivých metrík modelu a eliminovali tie, ktoré sa ukázali ako nevhodné. Otestovali sme detekčný mechanizmus navrhnutého systému pomocou datasetov *CSIC* a *Smihla* a výsledky sme porovnali s výsledkami systému PHPIDS. V porovnaní náš systém síce dosiahol vyšší počet FP vyhodnotili sme ho ako lepší najmä kvôli zlyhaniu systému

PHPIDS pri datasete *CSIC*. Z výsledkov je tiež zreteľný výrazný dopad ladiaceho modulu na celkový výsledok nášho systému. Na základe vykonaných testov teda môžeme konštatovať splnenie cieľov zadaných v sekcii 4.1.

V systéme sme identifikovali viacero oblastí, ktoré by mohli prispieť k zlepšeniu výstupu navrhnutého systému. Prvým je určenie váh typov signálov pre vytváranie kumulovaných výstupných signálov kde navrhujeme využitie metódy strojového učenia pre optimalizáciu kombinácie hodnôt. Ďalšou oblasťou je experimentovanie s nastaveniami ladiaceho evolučného algoritmu spolu s rôznymi metódami selekcie, mutácie a nahradenia.

Možné rozšírenie vidíme v implementácii viacerých agentov oboch druhov, a to najmä agenta typu zberač zameraného na analýzu logov konkrétnej webovej aplikácie. Ďalšou možnosťou je rozšírenie systému o detekčný mechanizmus na základe príznakov ktorý by pomohol znížiť FPR.

Literatúra

[1] ALSALEH, M., ALQAHTANI, A., ALARIFI, A., et al.: Visualizing PHPIDS Log Files for Better Understanding of Web Server Attacks. In *Proceedings of the Tenth Workshop on Visualization for Cyber Security*, VizSec '13, New York, NY, USA: ACM, 2013, ISBN 978-1-4503-2173-0, s. 1–8.

[2] BÄCK, T.: *Evolutionary Algorithms in Theory and Practice*. New York: Oxford University Press, 1995, ISBN 0195099710.

[3] BAIO., A.: Star Wars Kid: The Data Dump. `http://waxy.org/2008/05/star_wars_kid_the_data_dump/`, [navštívené 6.5.2016].

[4] BROWNLEE, J.: Dendritic Cell Algorithm. `http://www.cleveralgorithms.com/nature-inspired/immune/dca.html`, [navštívené 30.11.2015].

[5] CEPONIS, J., CEPONIENE, L., VENCKAUSKAS, A., et al.: Evaluation of Open Source Server-Side XSS Protection Solutions. In *Information and Software Technologies*, *Communications in Computer and Information Science*, ročník 403, editácia T. Skersys; R. Butleris; R. Butkiene, Springer Berlin Heidelberg, 2013, ISBN 978-3-642-41946-1, s. 345–356.

[6] CHAKRABARTI, S., CHAKRABORTY, M., MUKHOPADHYAY, I.: Study of Snort-based IDS. In *Proceedings of the International Conference and Workshop on Emerging Trends in Technology*, ICWET '10, New York, NY, USA: ACM, 2010, ISBN 978-1-60558-812-4, s. 43–47.

[7] CHAUHAN, P., CHANDRA, N.: Adopting Dendritic Cell Algorithm to Conform a Hybrid Intrusion Detection System. In *International Conference on Advances in Computer Science, AETACS*, editácia H. Prasad N.; N. N., 1, Elsevier Science Publishers B. V., 2013, ISSN 9789351071020, s. 364–372.

[8] DANFORTH, M.: WCIS: A Prototype for Detecting Zero-day Attacks in Web Server Requests. In *Proceedings of the 25th International Conference on Large Installation System Administration*, LISA'11, Berkeley, CA, USA: USENIX Association, 2011, s. 21–14.

[9] DASGUPTA, D., NINO, F.: *Immunological Computation: Theory and Applications*. Boston, MA, USA: Auerbach Publications, prvé vydanie, 2008, ISBN 1420065459, 9781420065459.

[10] DEBAR, H., DACIER, M., WESPI, A.: A revised taxonomy for intrusion-detection systems. *Annales Des Télécommunications*, ročník 55, č. 7-8, 2000: s. 361–378, ISSN 0003-4347.

[11] DUCH, W.: What Is Computational Intelligence and Where Is It Going? In *Challenges for Computational Intelligence*, *Studies in Computational Intelligence*, ročník 63, editácia W. Duch; J. Mańdziuk, Springer Berlin Heidelberg, 2007, ISBN 978-3-540-71983-0, s. 1–13.

[12] FENG, G., GREENSMITH, J., AICKLEIN, U.: The Dendritic Cell Algorithm for Intrusion Detection. In *Biologically Inspired Networking and Sensing: Algorithms and Architectures*, editácia P. Lio; D. Verma, IGI Global, 2012, ISBN 9781613500927, s. 84–102.

[13] GELBMANN, M.: WordPress powers 25% of all websites. `http://w3techs.com/blog/entry/wordpress-powers-25-percent-of-all-websites`, [navštívené 30.11.2015].

[14] GREENSMITH, J.: *The Dendritic Cell Algorithm*. Dizertačná práca, School of Computer Science, University Of Nottingham, 2007.

[15] GREENSMITH, J., AICKELIN, U.: Dendritic Cells for SYN Scan Detection. In *Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, GECCO '07, New York, NY, USA: ACM, 2007, ISBN 978-1-59593-697-4, s. 49–56.

[16] GREENSMITH, J., AICKELIN, U.: The Deterministic Dendritic Cell Algorithm. In *Artificial Immune Systems*, *Lecture Notes in Computer Science*, ročník 5132, editácia P. Bentley; D. Lee; S. Jung, Springer Berlin Heidelberg, 2008, ISBN 978-3-540-85071-7, s. 291–302.

[17] GREENSMITH, J., AICKELIN, U., CAYZER, S.: Detecting Danger: The Dendritic Cell Algorithm. In *Robust Intelligent Systems*, editácia A. Schuster, Springer London, 2008, ISBN 978-1-84800-260-9, s. 89–112.

[18] GREENSMITH, J., AICKELIN, U., TEDESCO, G.: Information Fusion for Anomaly Detection with the Dendritic Cell Algorithm. *Inf. Fusion*, ročník 11, č. 1, 2010: s. 21–34, ISSN 1566-2535.

[19] GREENSMITH, J., AICKELIN, U., TWYCROSS, J.: Articulation and clarification of the dendritic cell algorithm. In *Artificial Immune Systems*, Springer, 2006, s. 404–417.

[20] HUA, Y., TAO, L., XINLEI, H., et al.: A Survey of Artificial Immune System Based Intrusion Detection. *The Scientific World Journal*, ročník 2014, 2014: 156790.

[21] KIM, J., BENTLEY, P. J., AICKELIN, U., et al.: Immune system approaches to intrusion detection – a review. *Natural Computing*, ročník 6, č. 4, 2007: s. 413–466, ISSN 1567-7818.

[22] MACH, M.: *Evolučné algoritmy*. Košice: Elfa, 2009, ISBN 9788080861230.

[23] MCINERNEY, D.: HTTP POST analyzer in Python. `http://danmcinerney.org/http-post-analyzer-in-python/`, [navštívené 30.11.2015].

[24] OWASP Foundation: OWASP Top 10 - 2013. Technická správa, OWASP Foundation, 2013.

[25] SCULLY, P. M. D.: CSIC 2010 HTTP Dataset in CSV Format (for Weka Analysis). `http://users.aber.ac.uk/pds7/csic_dataset/csic2010http.html`, [navštívené 6.5.2016].

[26] SIGNH, N., PANDEY, Y.: Dendritic Cell Algorithm and Dempster Belief Theory Using Improved Intrusion Detection System. *International Journal of Advanced Research in Computer Science and Software Engineering*, ročník 3, č. 7, 2013: s. 699–703, ISSN 2277 128X.

[27] ŠMIHLA, Š.: HttpParamsDataset. `https://github.com/Morzeux/HttpParamsDataset`, [navštívené 6.5.2016].

[28] ŠOLTÉS, F.: *Evolučno-algoritmická syntéza hudobných prvkov*. Bakalárska práca, Fakulta informatiky a informačných technológií, Slovenská technická univerzita, 2014.

[29] TIMMIS, J., ANDREWS, P., OWENS, N., et al.: An interdisciplinary perspective on artificial immune systems. *Evolutionary Intelligence*, ročník 1, č. 1, 2008: s. 5–26, ISSN 1864-5909.

[30] TWYCROSS, J.: *Integrated Innate and Adaptive Artificial Immune Systems Applied to Process Anomaly Detection*. Dizertačná práca, School of Computer Science, University Of Nottingham, 2007.

[31] WALID M., A., MOHD N., O.: A detector generating algorithm for intrusion detection inspired by artificial immune system. *ARPN Journal of Engineering and Applied Sciences*, ročník 10, č. 2, 2015: s. 608–612, ISSN 1819-6608.

[32] WEISE, T.: *Global Optimization Algorithms – Theory and Application*. it-weise.de (publikované autorom): Germany, 2009.

[33] WU, X., S., BANZHAF, W.: The use of computational intelligence in intrusion detection systems: A review. *Applied Soft Computing*, ročník 10, č. 1, 2010: s. 1 – 35, ISSN 1568-4946.

Príloha A

Obsah elektronického média

K dokumentu priložené elektronické médium má nasledovnú štruktúru:

/doc/

- diplomový projekt 2 spolu s anotáciami v slovenskom a anglickom jazyku

/doc/latex/

- súbory dokumentácie vo formáte LaTeX

/src/

- zdrojové súbory implementovanej aplikácie

/ref/

- pdf súbory niektorých zdrojov

/README

- popis obsahu média v slovenskom a anglickom jazyku

Príloha B

Technická dokumentácia

B.1 Vysokoúrovňový opis implementácie

B.1.1 Klientská časť

Klientská časť systému je rozdelená do troch hlavných modulov:

1. core
2. analyzers
3. loaders

Na obrázku B.1 je zobrazený diagram tried pre vybranú množinu tried implementácie klientskej časti systému.

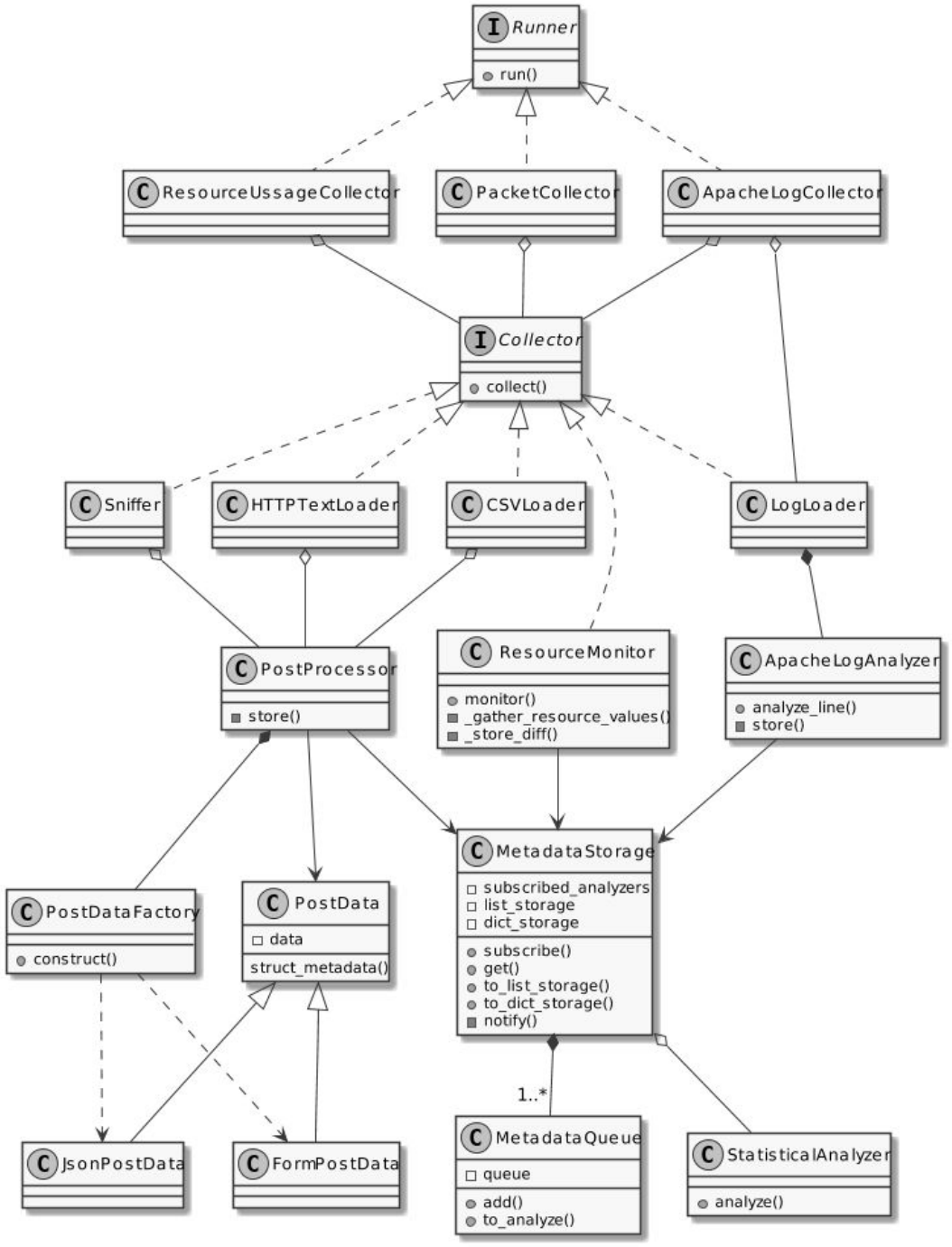
core

Modul *core* obsahuje submodul pre úložisko metadát a modul pre komunikáciu so serverovou časťou. Úložisko metadát je koncipované ako zdieľaný objekt, ktorý obsahuje dátovú štruktúru úložiska. Hlavnou časťou rozhrania objektu sú metódy pre vloženie dát do úložiska. Keďže úložisko metadát funguje ako *subject*, v návrhovom vzore *observer* obsahuje metódy pre registráciu a pre notifikáciu jednotlivých objektov typu *observer*.

Rozhranie modulu pre komunikáciu so serverom obsahuje jedinú metódu, ktorá odošle signál vstupujúci ako parameter na serverovú časť.

loaders

Modul *loaders* sa skladá zo submodulov jednotlivých zberačov. Modul je implementovaný pre čo najvyššiu rozšíriteľnosť. Pre pridanie nového zberača stačí implementovať vyžadované rozhranie a následne pridať meno nového submodulu do konfiguračného súboru.



Obr. B.1: Diagram tried vybranej množiny tried implementácie klientskej časti systému.

Každý zberač musí byť implementovaný ako trieda vlákna, pričom musí obsahovať metódu *run*, ktorá vlákno spustí. Modul tiež musí exportovať štruktúru *signal_mapping* ktorá obsahuje mená jednotlivých signálov modulu spolu s ich mapovaním a mierou škálovania. Vlákno priamo využíva zdieľaný objekt *storage* z modulu *core* pre ukladanie dát do úložiska. Interná štruktúra zberača nie je pre funkcionalitu systému dôležitá.

V našej implementácii sú zberače zložené z troch elementov:

analyzer Úlohou modulu je agregácia dát a plnenie úložiska. Je potrebné predchádzať zámene s modulmi typu analyzátor.

collector Má za úlohu zber dát zo zdroja a normalizáciu týchto dát do príslušných dátových štruktúr

runner V module sa nachádzajú konštruktory jednotlivých objektov a ich korektné prepojenie.

analyzers

Modul *analyzers* obsahuje submoduly, ktoré slúžia na analýzu dát v úložisku metadát a ich transformáciu na signály. Každý modul typu analyzátor je implementovaný ako *observer* a musí obsahovať metódu *analyze*, ktorá je spúšťaná úložiskom. Priamo analyzátorom je tiež spúšťaná metóda pre poslanie dát na serverovú časť z modulu *core*.

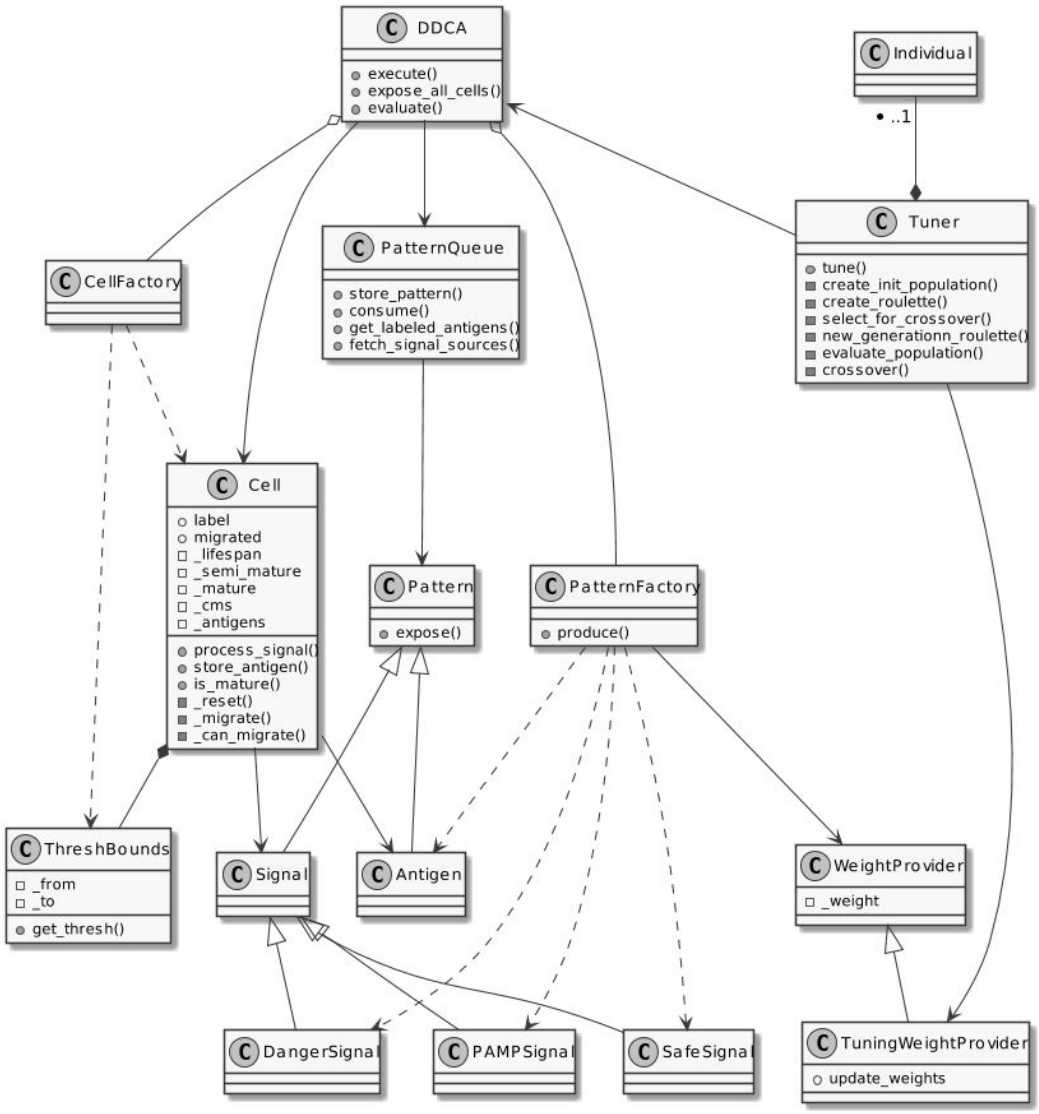
B.1.2 Serverová časť

Serverová časť systému je rozdelená do troch hlavných modulov:

1. core
2. storage
3. api

Dôležitou časťou systému je tiež modul *commands*, ktorý slúži ako rozhranie pre spúšťanie detekcie a ladenia.

Na obrázku B.2 je zobrazený diagram tried pre vybranú množinu tried implementácie serverovej časti systému.



Obr. B.2: Diagram tried vybranej množiny tried implementácie serverovej časti systému.

core

Modul *core* obsahuje triedy potrebné pre algoritmus *dDCA* a ladiaci evolučný algoritmus. Triedy algoritmu sú sústredené v submodule *pattern*, ktorý obsahuje špecifikáciu jednotlivých typov signálov. Submodul *cell* obsahuje abstrakciu dendritickej bunky a submodul *ddca* obsahuje vytvorenie potrebných obektov a samotnú logiku algoritmu. Ladenie je sústredené v submodule *tuner*.

storage

Modul *storage* slúži pre umiestnenie modulov vytvárajúcich abstrakciu nad databázou. Bol implementovaný kvôli tomu, aby sme znížili previazanosť našej implementácie s konkrétnou databázou. Modul obsahuje metódy pre ukladanie a získavanie signálov. Taktiež obsahuje triedu pre správu váh jednotlivých signálov z fázy ladenia.

api

Modul *api* obsahuje moduly potrebné pre vytvorenie jednoduchého webového servera využitím rámca *Flask*. Webový server obsahuje jediný koncový bod určený na uloženie vzorky. V module sa tiež nachádza submodul pre validáciu vstupných premenných.

B.2 Inštalácia systému

Systém je implementovaný pre operačné systémy GNU/Linux. Pre fungovanie systému je potrebná inštalácia interpretera *Python* vo verzii 3 a nástroja *pip*. Zoznam závislostí sa nachádza v súbore requiremets.txt, ktorý je umiestnený v adresári servera a klienta. Závislosti je možné nainštalovať príkazom

```
$ pip install -r requirements.txt
```

zvlášť pre klientskú a serverovú časť. Pre fungovanie serverovej časti systému je potrebné nainštalovať databázu *MongoDb*.

B.3 Používateľská príručka

Pri serverovej časti je potrebné spustiť webové api pomocou príkazu:

```
$ python web.py
```

Analýzu je možné spúšťat pomocou príkazu:

```
$ python commands.py dca
```

Pri tomto príkaze je použitá konfigurácia nachádzajúca sa v databáze. Ladenie je možné spustiť príkazom:

```
$ python commands.py tune
```

Po dokončení ladenia je finálna konfigurácia zapísaná do databázy.

Klientskú časť je možné spustiť príkazom

```
$ sudo python run.py
```

Práva administrátora sú potrebné kvôli zachytávaniu dát priamo z paketov. Po spustení začne systém monitorovať definované zdroje údajov.

Príloha C

Návrh vedeckého článku

Use of a Dendritic Cell Algorithm in Web System Anomaly Detection

Filip ŠOLTÉS*

Faculty of Informatics and Information Technologies STU Bratislava
f@fslts.org

Abstract. Many web systems currently contain vulnerabilities which could be easily misused by attackers, and consequences of this misuse can be fatal. In this paper we propose a system that improves the security of a web system by anomaly detection. Detection is achieved using a combination of an artificial immune system (AIS) and an evolutionary algorithm (EA). The use of AIS seems proper in this context since attacks on web systems could be seen as attacks of pathogens on an organism. Our system collects and analyzes data from monitored web system using specialized agents. Gathered data is then evaluated with one of the AIS algorithms which is the dendritic cell algorithm. We are modifying the algorithm with a tuning module in which we use EA for the optimization weight of data from specific sources.

1 Introduction and related work

With the expansion of web systems that we can see nowadays, we often see attempts of intrusion in these systems. There are multiple options for securing users, data, or systems themselves, such as firewalls or encryption. One of the more complex security measures are intrusion detection systems.

There are multiple approaches for intrusion detection systems, one part of which are biology inspired methods. In further sections we present three existing intrusion detection systems and then description of design and evaluation of our own intrusion detection system that uses biology inspired methods.

Today there are many commercial or non-commercial IDS options. In further section we present three systems - *Snort*, *PHPIDS* and *WCIS*. Each of these systems use different detection approaches.

Snort is a popular system for intrusion detection which uses flag-based detection mechanism. It is widely used mainly for network traffic inspection and it is rated as one of the best solutions available. A set of rules are used for detection. These rules are stored in text files for easy modification. Snort contains a great amount of predefined rules which detect intrusion attempts. Application is logically split into several components (packet decoder, preprocessors, detection mechanism, logging system,

---

* Master study programme in field: Software Engineering
Supervisor: Assoc. Professor Ladislav Hudec, Institute of Applied Informatics, Faculty of Informatics and Information Technologies STU in Bratislava

output modules). These components cooperate in the detection of specific attacks and in output generation.

PHPIDS is free IDS oriented to PHP applications [1]. By inspection of input variables in POST and GET requests and HTTP cookie can PHPIDS detect most of the conventional attacks such as XSS, SQL injection, DoS or path traversal.

PHPIDS uses a detection method based on flags and uses predefined samples. The system also contains components for deep string and input metrics analysis by which it is possible to detect unknown attacks [1].

WCIS is a system focused on the detection of unknown attacks and is designed as AIS enhanced with request classification [2]. Classification is done by *URI* analysis. Six types of attacks are defined in a system for which sensors are created using normal and attack datasets. For individual sensors training and maturation is needed. This is achieved in the typical life cycle of a negative selection algorithm. Data abstraction was created using a number of parameters, length of *URI*, etc.

During training phase sensors were created and evaluated. Each sensor was assigned one of the six attacks based on its detection capability. In the next phase a genetic algorithm was used. Its goal was the optimization of detection capability of individual sensors.

2 Our approach

For our system we defined three main goals: 1) the system will be applicable in real size web systems; 2) minimization of resource consumption on monitored system; and 3) minimization of required user interaction. With consideration of these goals, studied solutions, and architecture suggestions in [6] we designed an IDS which detects anomalies of a web system using computing intelligence. In particular, we have chosen a dendritic cell algorithm in combination with evolutionary algorithm. The solution is separated into a client side and a server side and each part we present in more detail in following sections.

2.1 Server side

The server side of the proposed system is used for processing data obtained from the client side. This data is in the form of *signals* and *antigens*, which are aggregated a data on high level of abstraction. The signals represent structural features of the system and antigens represent behavioral features of the system. Anomaly detection of events is made based on the results of signal and antigen processing. The architecture of the server side is shown in Figure 1. The core component of the server side is AIS implemented in a form of the deterministic dendritic cell algorithm (dDCA). By use of deterministic form of the algorithm we wanted to simplify evaluation of other parts of the design. A detailed description of deterministic dendritic cell algorithm is provided by authors in [4].

Original dDCA design does not contain any form of machine learning [3]. In our design we have decided to enrich dDCA by a form of tuning for which we picked an evolutionary algorithm. As a primary point of optimalization we picked the weights of input signals by their source. An individual is defined as set of weights for each of the sources present in the system and we can represent it with a binary vector of the size σ. This size is defined by formula 1.

$$\sigma = \lambda * \lceil \log_2 (\theta) \rceil \tag{1}$$

Symbol λ represents the number of sources in the tuning process. Position of the source in the vector is defined by alphabetic order of the code names of metrics. Symbol θ represents number of

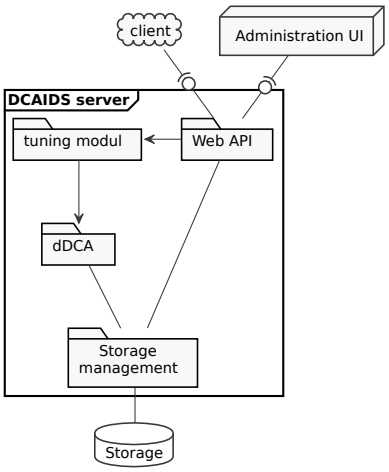


Figure 1. Architecture of the server side.

different weight values. The way to convert the weight identifier θ_x to the weight value of the source is described in formula 2.

$$w(p) = \begin{cases} \frac{1}{\theta_x} + 1, & \text{if } \theta_x \leq \frac{\theta}{2} \\ \theta_x + 1, & \text{otherwise} \end{cases} \tag{2}$$

In the formula $w(p)$ is the weight of the source of the individual x, θ_x is the specific weight identifier of the source p for the individual x.

The fitness function is formalized in the formula 3 where x represents a individual, $\alpha(x)$ represents true positive rate for the individual x and $\beta(x)$ represents false positive rate for the individual x. Symbol c is an constant that defines rate of penalisation of false positives.

$$f(x) = \alpha(x) - c\beta(x) \tag{3}$$

2.2 Client side

The client side of the system is used for data collection, aggregation, and analysis. Analyzed data is transformed to signals and antigens and sent to the server side. The architecture of server side is shown in Figure 2.

As an antigen we propose an IP address of an HTTP request since, to a certain extent, we can connect an action with a single user. The application consists of two types of agents: *gatherers* and *analyzers*. *Gatherers* are used for data collection from different sources and there is a single *gatherer* for every data source, of which, at the moment, there are three:

- web server logs,
- HTTP POST data,
- system resources usage.

Collected data is aggregated and put into common storage. From the web server logs we store a number of requests per time interval and the number of unique IP addresses of requests per time

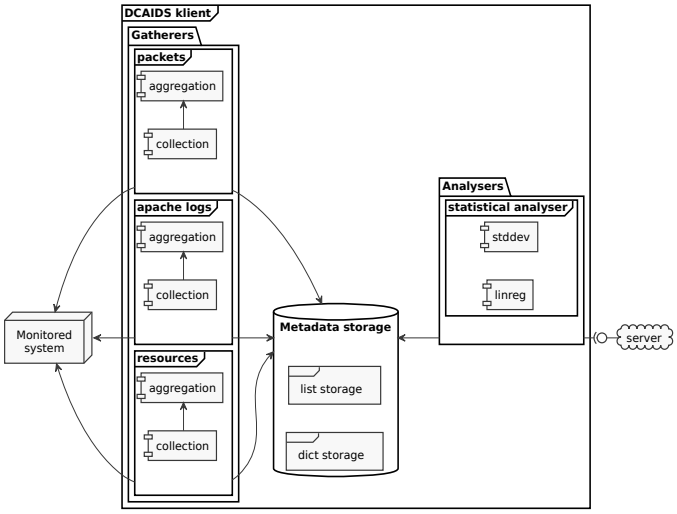


Figure 2. Architecture of the client side.

interval. The intended goal of proposed metrics is to detect attempts of automated scanning of vulnerabilities. Another part is the storing count of error events per time interval since it is a direct sign of an anomaly.

The abstraction of HTTP POST data is created mainly for security reasons so we won't store any sensitive data sent by the user. We store three values constructed from strings:

1. number of alphanumeric characters,
2. number of punctuation characters,
3. number of other characters,

From these values we compute the ratio of punctuation characters to alphanumeric characters and the ratio of other characters to alphanumeric characters. We have chosen these values as we assume that malicious requests reach different values, since they often use special characters from languages as *SQL*. Using received values we can monitor the rate of occurrence of individual parameters and detect invalid parameters.

For system resources we observe:

- CPU (immediate usage),
- memory (immediate usage),
- network interface (number of received/sent packets, number of received/sent bytes, number of errors during sent/receive, number of dropped packets that should be received/sent),
- disc (number of read/writes, number of read/write bytes, time spent with read/write).

For data exchange between modules we propose simple storage which, is composed of a set of vectors. The set is organized by unique keys which define the data origin and meaning of data in the vectors. Every vector represents a front with limited size.

Every analyzer loads and analyses the whole content of every vector from the storage. Analysis is done in the same way for every vector regardless its source. The purpose of the analysis is to

Table 1. Parameters of systems used for evaluation.

CPU	Intel(R) Core(TM) i7-4600U CPU @ 2.10GHz
CPU cores	1
RAM	1.5GB
OS	CentOS Linux release 7.1.1503 (core)
Virtualisation environment	VirtualBox 5.0.10_OSEr104061

Table 2. Average values of metrics of client side resource impact on test system.

	CPU usage (%)	Response time (ms)
Without IDS	73.9560	1135.1645
With IDS	73.4491	1165.2988

create input for the server side in the form of signals. We represent signals as algorithm authors in [5]. Every signal represents real number value metrics normalized to the interval $< 0, 100 >$. For each metric a mapping and scaling was performed. By this we mean the assignment of metrics to a particular type of signal and scaling of the value to the mentioned interval.

In the current state of the system we propose one analyzer used for statistical analysis. For every vector the analyzer computes thus: maximal/minimal value of vector, standard deviation, linear regression, average value of vector and median of vector.

3 Evaluation

For the purposes of evaluation we have created an implementation of the server and client side. Both the client and server side are implemented in python; the server side is implemented as a web application in the framework *Flask*.

Since the minimization of resource consumption on monitored systems was one of the defined goals, we identified the evaluation of the client side impact on monitored system as a priority. We have chosen two metrics: CPU usage and response time to HTTP request. Our goal is to watch these metrics during the increased load on a web server. We have deployed a web server and installed the application *Wordpress* on a virtual machine whose parameters are shown in the table 1.

A test was performed using the *ab* tool by which 10000 HTTP requests were generated in 10 concurrent sessions. This scenario was repeated twice, with and without IDS deployed. The tool *ab* was also used to capture the response time of the web server. CPU usage was captured using the tool *dstat*. Results of the test are shown in the table 2, and we see that the implementation of the client side has a negligible impact from the viewpoint of CPU usage and response time to HTTP requests.

We also evaluated the detection mechanism of our system with dataset *CSIC 2010*. The test was done by resending the requests in the dataset to the test PHP application. We repeated the test twice for our system and the PHPIDS system. Since there are no timing information in the dataset, we sent the requests in random intervals with average of a hundred requests per second. We evaluated true positive rate (TPR) and false positive rate (FPR) for both systems. We did the evaluation for two configurations of weights of particular signal types for creating cumulated output signals summarized in table 3. We also did the evaluation with and without the results of tuning module.

Results of the evaluation are shown in table 4 where out system is marked as DCAIDS. We can see, that FPR of our system is higher, but TPR is also significantly higher which in our opinion justifies the FPR value. We can also see, that the tuning module recognizably refined the achieved results.

Table 3. Weights of particular signal types for creating cumulated output signals..

Signal	Configuration	PAMP signal	Danger signal	Safe signal
CMS	1	3	1	7
semi-mature DC		0	0	3
mature DC		9	5	-3
CMS	2	3	1	7
semi-mature DC		0	0	3
mature DC		9	3	-4

Table 4. Results of the final test of the system and comparison with PHPIDS for dataset CSIC 2010*.*

System	DCAIDS				PHPIDS
Configuration	1		2		-
Tuning	✗	✓	✗	✓	-
TPR	48.9990	64.9189	81.9157	87.8811	15.6034
FPR	6.9291	6.2723	14.4035	12.7115	0.5555

4 Conclusion and future work

In this paper we presented our design of an intrusion detection system for improving web system security using biology-inspired methods. In our design we use the deterministic dendritic cell algorithm for anomaly detection and an evolutionary algorithm for tuning. The main portion of our work was in defining metrics that should be monitored on the web system.

We have implemented and evaluated a prototype. In the first presented test we tested the implementation itself and its resource consumption in the monitored system. In the second presented test we evaluated the detection mechanism of our system in which shown significantly higher value of TPR which justifies moderately higher FPR value.

As a main task for future work we see a further tuning if the defined parameters since we see an possibility for further refinement of the results.

Acknowledgement: This work was partly sponsored by the Eset Research Centre and Scientific Grant Agency of the Slovak Republic, grant No. VEGA 1/0836/16 Methods and algorithms for improving efficiency and multimedia content delivery in IP networks.

References

[1] Alsaleh, M., Alqahtani, A., Alarifi, A., Al-Salman, A.: Visualizing PHPIDS Log Files for Better Understanding of Web Server Attacks. In: *Proceedings of the Tenth Workshop on Visualization for Cyber Security*. VizSec '13, New York, NY, USA, ACM, 2013, pp. 1–8.

[2] Danforth, M.: WCIS: A Prototype for Detecting Zero-day Attacks in Web Server Requests. In: *Proceedings of the 25th International Conference on Large Installation System Administration*. LISA'11, Berkeley, CA, USA, USENIX Association, 2011, pp. 21–14.

[3] Greensmith, J.: *The Dendritic Cell Algorithm*. PhD thesis, School of Computer Science, University Of Nottingham, 2007.

[4] Greensmith, J., Aickelin, U.: The Deterministic Dendritic Cell Algorithm. In Bentley, P., Lee, D., Jung, S., eds.: *Artificial Immune Systems*. Volume 5132 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2008, pp. 291–302.

[5] Greensmith, J., Aickelin, U., Twycross, J.: Articulation and clarification of the dendritic cell algorithm. In: *Artificial Immune Systems*. Springer, 2006, pp. 404–417.

[6] Kim, J., Bentley, P.J., Aickelin, U., Greensmith, J., Tedesco, G., Twycross, J.: Immune system approaches to intrusion detection – a review. *Natural Computing*, 2007, vol. 6, no. 4, pp. 413–466.