VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
PROGRAMŲ SISTEMŲ STUDIJŲ PROGRAMA

GAN architektūrų, tinkamų kenkėjiško kodo obfuskacijai, analizė

Analysis of GAN architectures suitable for Ethical Malware Obfuscation

Kursinis darbas

Atliko: 4 kurso 3 grupės studentas
Liudas Kasperavičius

Darbo vadovas: prof. dr. Olga Kurasova

Vilnius – 2024

Turinys

Įvadas

Pastaraisiais metais kenkėjiškas kodas ir programos kuriamos itin sparčiai (∼450000 kenkėjiškų programų per dieną AV-TEST duomenimis). Kenkėjiško kodo aptikimo programos, kurios tradiciškai remiasi programų pėdsakais (*angl. signature*), nespėja atnaujinti pėdsakų duomenų bazių pakankamai greitai. Dėl to Dirbtinio Intelekto (DI), tiksliau Mašininio Mokymosi (ML), naudojimas kenkėjiškų programų ar kenkėjiško kodo aptikimo srityje tapo itin populiarus [DCB+21]. Tačiau ML modeliai, nors ir geba aptikti kenkėjiškas programas iš naujų, dar nematytų, duomenų, yra pažeidžiami varžymosi principais pagrįstoms atakoms (*angl. adversarial attacks*) [CSD19; HT17; RSR+18; ZHZ+22]. Šių atakų principas yra ML modelio sprendimų priėmimo ribos (*angl. decision boundary*) radimas – žinant šią ribą pakanka pakeisti kenkėjiškos programos veikimą taip, kad ML modelis priimtų sprendimą klasifikuoti ją kaip nekenksmingą [DCB+21]. Žinoma, rasti šią ribą nėra trivialus uždavinys. Mokslinėje literatūroje išskiriami 3 ribos paieškos atvejai [FWL+19]:

1. **Baltos dėžės** atvejis: kenkėjiško kodo kūrėjas turi visą informaciją apie ML modelį, t. y. modelio architektūrą, svorius, hiperparametrus.
2. **Juodos dėžės su pasitikėjimo įverčiu** atvejis: kenkėjiško kodo kūrėjas gali tik testuoti modelį – t. y. pateikti programą ir gauti atsakymą. Atsakymo forma – klasifikacija ir tikimybė, kad klasifikacija yra teisinga (pasitikėjimo įvertis).
3. **Juodos dėžės** atvejis: kenkėjiško kodo kūrėjas gali tik testuoti modelį. Atsakymo forma yra tik klasifikacija.

Akivaizdu, jog „juodos dėžės“ atvejis yra sudėtingiausias, bet ir labiausiai atitinka realias sąlygas **[CITE]**. Todėl šiame darbe nagrinėjami modeliai, gebantys generuoti varžymosi principais pagrįstų atakų obfuskuotus kenkėjiško kodo pavyzdžius (AE) „juodos dėžės“ atvejams.

Tikslas – nustatyti labiausiai tinkantį modelį varžymosi principais pagrįstoms atakoms „juodos dėžės“ atvejais.

Uždaviniai:

1. Apžvelgti kenkėjiško kodo obfuskacijos metodus
2. Nustatyti kriterijus varžymosi principais grįstų atakų modeliams ir juos įvertinti
3. Atlikti eksperimentinį tyrimą naudojant modelį, gavusį aukštą įvertinimą pagal kriterijus

1. Literatūros apžvalga

Mokslinėje literatūroje kenkėjiško kodo ir programų obfuskacijai daugiausia dėmesio skiriama AE generavimui, dėl paplitusio ML modelių naudojimo kenkėjiško kodo ir programų detekcijos srityje **[CITE]**. Dažniausia atakų platforma – operacinė sistema „Windows“ ir jos naudojami PE formato failai **[CITE]**.

Kenkėjiškų programų analizė gali būti išskiriama į **statinę analizę** ir **elgesio (*angl. behavior*) analizę**. Statinės analizės detektoriai yra labiau paplitę ir prieinami, kadangi požymių išskyrimas iš tam tikro iš anksto žinomo formato failų ir jų klasifikavimas yra sąlyginai nesudėtinga užduotis. Tuo tarpu elgesio analizės detektoriai dažniausiai neprieinami eiliniams vartotojams, o diegiami korporacijose **[CITE]**. Jų veikimas pagrįstas nuolatiniu sistemos stebėjimu ir anomalijų detekcija, tad yra kartu ir sudėtingesnė, ir reikalaujanti daugiau resursų, nei statinė analizė, užduotis.

Mokslinėje literatūroje nagrinėjant specifinį būdą generuoti AE, apibrėžiamas varžymosi principais pagrįstų atakų karkasas. Karkasą sudaro

- Surogatinio modelio tipas ir architektūra
- ML modelio tipas ir architektūra
- ML modelio naudojami požymiai (žr. 1.1)
- ML modelio naudojamos perturbacijos (žr. 1.2)

1.1. Naudojami požymiai

Varžymosi principais pagrįstos atakos taikosi į ML modeliais paremtus kenkėjiškų programų detektorius. Šie detektoriai yra klasifikatoriai – pateiktį (programą) klasifikuoja kaip kenkėjišką (*angl. malicious*) arba nekenkėjišką (*angl. benign*). Kadangi programos nėra fiksuoto dydžio, klasifikatoriai remiasi programų požymiais, kurie gaunami atliekant požymių ištraukimą (*angl. feature extraction*). Laikoma, jog „juodos dėžės“ atvejais sužinoti kokius tiksliai požymius vertina kenkėjiškų programų detektorius yra neįmanoma, tad karkasų apibrėžimuose, priklausomai nuo jų specifikos ir tikslų, neretai pateikiami jų vertinami programų požymiai. Šioje sekcijoje išskiriami ir klasifikuojami mokslinėje literatūroje minimi požymiai.

1.1.1. PE formato programų požymiai

- **DLL vardai (arba API vardai [HT17])** [ZCY+24]. PE faile turi būti nurodyti visi naudojami DLL. Prieš pradedant treniruoti ML modelį, atliekama visų turimų programų analizė ir nustatoma visų naudojamų DLL aibė D. Tarkime $|D| = n$. Tuomet, požymių vektorius programai, naudojančiai $X \subseteq D$ DLL, bus n-matis dvejetainis vektorius, kurio i-asis elementas yra $\begin{cases} 0, \text{ jei } D_i \notin X, \\ 1, \text{ jei } D_i \in X. \end{cases}$ Čia D_i – i-asis D elementas.
- **PE metaduomenys** [AKF+18]. Tai visi PE formato faile esantys metaduomenys, tokie, kaip sekcijų pavadinimai, sekcijų dydžiai, „ImportTable“ ir „ExportTable“ metaduomenys ir kt. Formuojant požymių vektorių skaičiuojama metaduomenų maišymo funkcija.

1.1.2. Baitų lygio požymiai (gali būti ištraukiami iš bet kokio formato failų)

- **Prasmingų žodžių (angl. Strings) kiekis** [AKF+18]. Prasmingus žodžius suprantame kaip turinčius prasmę žmogui *(angl. human readable)*. Tai gali būti URL, failų keliai *(angl. file paths)* ar registro raktų pavadinimai. Kadangi prasmingų žodžių kiekis tėra vienas skaičius, požymių vektorius dažniausiai formuojamas prijungiant ir kitus požymius.
- **Baitų/entropijos histograma** [SB15]. Specifinis metodas, užkoduojantis dažniausiai pasikartojančias baitų ir entropijos poras 256 dimensijų vektoriumi.
- n**-gramos** [ZZY+22]. Dažniausiai sutinkamos skaitmeniniame natūraliosios kalbos apdorojime (NLP). Tai yra n žodžių junginiai, arba, sukompiliuotų programų apdorojimo kontekste, n baitų junginiai. Nustatant požymių vektorių, visos n-gramos surikiuojamos pagal pasikartojimą programoje mažėjimo tvarka („populiariausios“ viršuje). Iš pirmų m reikšmių sudaromas m-matis vektorius – tai ir yra požymių vektorius.

1.2. Perturbacijos

Perturbacijos – tai pagrindinis obfuskacijos metodas AE kūrimui. Perturbacijų tikslas yra pakeisti kenkėjiškos programos veikimą išsaugant originalų funkcionalumą. Perturbacijos gali būti sudėtingos ir apimti visą programą (pvz. visos programos užšifravimas ir pridėjimas prie kitos programos), semantinės (pvz. tam tikrų mašininio kodo instrukcijų keitimas į ekvivalentų rezultatą pasiekiančias) arba baitų lygio (pvz. nulinių baitų pridėjimas programos gale) **[CITE]**. Perturbacijų parinkimas įeina į karkaso apibrėžimą. Šioje sekcijoje aptariamos mokslinėje literatūroje minimos perturbacijos.

1.2.1. Baitų lygio perturbacijos

- **„ARBE“ (*Append Random Bytes at the End*)** [FWL+19]. PE formato failo gale pridedami atsitiktiniai baitai.
- **„ARI“ (*Append Random Import*)** [FWL+19]. PE formato failo „ImportAddressTable“ lentelėje pridedama atsitiktinai pavadinta biblioteka su atsitiktinai pavadinta funkcija.
- **„ARS“ (*Append Randomly named Section*)** [FWL+19]. PE formato failo „SectionTable“ lentelėje pridedamos atsitiktinės sekcijos (sekcijos ir jų tipai yra apibrėžti PE formate).
- **„RS“ (*Remove Signature*)** [FWL+19]. Sertifikato pašalinimas iš PE formato failo „CertificateTable“ lentelės.
- **Naujas įeities taškas** [AKF+18]. Prasidėjus programai, iškart peršokama nuo naujo įeities taško į originalųjį.
- **„Header Fields“** [DCB+21]. PE formato failo „PE Header“ ir „Optional Header“ dalių specifinių laukų keitimas (pvz. sekcijos pavadinimo keitimas [AKF+18]).
- **„Partial DOS“** [DCB+21]. PE formato failo „DOS Header“ dalies pirmi 58 baitai po „MZ“ skaičiaus yra nenaudojami moderniose operacinėse sistemose, tad juos galima keisti.
- **„Slack Space“** [DCB+21]. Dėl PE formato specifikos, kiekviena nauja sekcija turi prasidėti tam tikro skaičiaus, nurodyto „PE Header“ dalyje, kartotiniu nuo pradžios. Kompiliatoriai

šį reikalavimą išpildo sekcijų gale pridėdami tiek nulinių baitų, kiek reikia teisingam sulygiavimui pasiekti. Būtent ši nulinių baitų erdvė gali būti keičiama be jokios įtakos originaliai programai.

- **„Padding“** [DCB+21]. Nulinių baitų pridėjimas failo gale.
- **„Full DOS“** [DCB+21]. Perturbacijos esmė tokia pat, kaip ir „Partial DOS“, tik naudojami visi „DOS“ dalies baitai, išskyrus „MZ“ ir „PE Offset“ („Partial DOS“ manipuliacijoms naudoja tik dalį tarp „MZ“ ir „PE Offset“).
- **„Extend“** [DCB+21]. Pakeičiama PE formato faile „DOS“ dalyje esanti „PE Offset“ reikšmė į didesnę[1]. Taip padidinama (išplečiama) visa „DOS“ dalis. Tolesnis perturbacijos principas yra toks pat, kaip ir „Full DOS“.
- **„Shift“** [DCB+21]. PE formato failuose kiekvienas sekcijos blokas prasideda su sekcijos vieta nuo pradžios (*angl. offset*). Tarkime ši reikšmė yra S. Sekcijos kodas pradedamas vykdyti tik nuo adreso $P + S$, kur P – programos pradžios adresas. Vadinasi, padidinus[1] S per n, atsiranda n baitų laisvos vietos iki sekcijos pradžios, kurią galima keisti be jokios įtakos programos veikimui.

1.2.2. Semantinės perturbacijos

- **Nereikalingų DLL/API vardų požymių pridėjimas** [HT17]. PE formato faile „ImportTable“ lentelėje pridedami originalios programos nenaudojami DLL/API vardai.
- **„Binary Rewriting“** [DCB+21]. Semantinis instrukcijų perrašymas. Pavyzdžiui, $A + B$ instrukcijos pakeitimas į $A - (-B)$.

1.2.3. Kompleksinės perturbacijos

- **„Obfusmal“** [ZCY+24]. Užšifruojama originalios programos kodo sekcija. Sukuriama ir originalios programos gale pridedama programa *Shell.dll*, kurioje laikomas atšifravimo raktas, originalios programos kodo sekcijos adresas ir dydis. Be to, *Shell.dll* geba atšifruoti originalios programos kodo sekciją ir jai perduoti kontrolę. *Shell.dll* pridedama prie naudojamų DLL, o programos pradžios taškas nustatomas į *Shell.dll* pradžios tašką. Iliustracija pateikiama 1-ame pav.
- **„Stealmal“** [ZCY+24]. Visa originali programa užšifruojama ir pridedama prie programos *Shell.exe* galo. *Shell.exe* geba atšifruoti originalią programą ir perduoti jai kontrolę. Iliustracija pateikiama 1-ame pav.
- **„Hollowmal“** [ZCY+24]. Užšifruojama visa originali programa. Ji pridedama prie kurios nors nekenksmingos programos galo. Prie šio junginio galo pridedama *Hollow.dll* programa, kurios veikiamas panašus į *Shell.exe* iš „Stealmal“. Viso junginio pradžios taškas nustatomas į *Hollowmal.dll* pradžios tašką. Iliustracija pateikiama 1-ame pav.

[1] šios reikšmės padidinimas reiškia visos failo struktūros keitimą („DOS“ dalis yra failo pradžioje). Būtina pakeisti visų sekcijų vietas nuo pradžios (*angl. offset*) jų metaduomenyse.

1.3. GAN tipo modelių karkasai

GAN modeliai paremti Generatyviniais Priešiškais Tinklais (*angl. Generative Adversarial Networks*), kurių veikimo principas yra du neuroniniai tinklai (generatorius ir diskriminatorius), žaidžiantys nulinės sumos žaidimą (*angl. zero-sum game*) **[CITE]**. Kenkėjiško kodo obfuskacijos kontekste ir ypač „juodos dėžės“ atvejais, diskriminatorius atlieka surogatinio modelio vaidmenį. Bendras GAN modelių mokymosi etapas yra tokia seka:

1. Generatorius, naudodamas požymių vektorių ir tokios pačios dimensijos „triukšmo“ (*angl. noise*) vektorių, sugeneruoja perturbacijas.
2. Originali kenkėjiška programa modifikuojama pagal perturbacijas (sukuriamas AE).
3. Diskriminatorius bando klasifikuoti sugeneruotą AE (kenkėjiškas / nekenkėjiškas). Diskriminatoriaus klasifikacija lyginama su tikro detektoriaus klasifikacija. Jei ji teisinga – atnaujinami generatoriaus parametrai pagal generatoriaus nuostolių funkciją. Kitu atveju, atnaujinami diskriminatoriaus parametrai pagal diskriminatoriaus nuostolių funkciją.
4. Visa seka kartojama nustatytą kiekį kartų.

[CITE].

1.3.1. „MalGAN“

Tikslas	Efektyviai išvengti AE aptikimo, kai ML kenkėjiškų programų detektoriaus implementacija nežinoma („juodos dėžės“ atvejis)
Surogatinis modelis	Daugiasluoksnis tiesioginio sklidimo neuroninis tinklas – klasifikatorius. Įvestis – programos požymių vektorius. Išvestis – klasifikacija į kenksmingą arba nekenksmingą. Šis tinklas taip pat naudojamas kaip diskriminatorius GAN architektūroje.
ML modelis	Daugiasluoksnis tiesioginio sklidimo neuroninis tinklas. Įvestis – programos požymių vektorius ir tokios pačios dimensijos „triukšmo“ vektorius. Išvestis – modifikuotas požymių vektorius. Šis tinklas naudojamas kaip generatorius GAN architektūroje.
Požymiai	• „MalGan“ straipsnyje naudojami tik API vardų požymiai, patenkantys į PE formato programų požymių kategoriją (žr. 1.1.1), tačiau autoriai nurodo, jog gali būti naudojami bet kokie požymiai.
Perturbacijos	• Semantinės perturbacijos (1.2.2) – nereikalingų API vardų požymių pridėjimas

[HT17]

1.4. Skatinamojo mokymosi tipo modelių karkasai

Skatinamojo mokymosi (RL) modeliai susideda iš agento ir aplinkos. Aplinka susideda iš informatyvių požymių ištraukimo metodo (*angl. feature extraction*) ir kenkėjiškų programų detektoriaus. Agentas – tai algoritmas ar neuroninis tinklas, kurio tikslas yra surasti optimalią strategiją (*angl. policy*). Šiuo atveju strategija susideda iš perturbacijų (žr. 1.2) **[CITE]**. Bendras RL modelių mokymosi etapas yra tokia seka:

1. agentas, naudodamas dabartinę aplinkos būseną ir praeito veiksmo atlygį (*angl. reward*), parenka sekantį veiksmą iš galimų veiksmų aibės
2. atliekamas veiksmas – perturbuojama programa arba požymių vektorius (priklauso nuo karkaso)
3. gaunami aplinkos kitimo įverčiai – nauja būsena ir atlygis, skaičiuojamas pagal detektoriaus klasifikacijos rezultatą
4. seka kartojama tol, kol agentas nelaiko strategijos optimalia arba nustatytą kiekį kartų

[CITE]

1.5. Genetinių algoritmų tipo modelių karkasai

Genetinai algoritmai (GA) yra viena seniausių mašininio mokymosi apraiškų; jų veikimas paremtas evoliucija **[CITE]**. Kenkėjiškų programų obfuskacijai AE generavimas taikant GA yra tokia seka:

1. sukuriama pradinė populiacija (perturbacijos metodai pradinei populiacijai priklauso nuo karkaso)
2. atliekamas vertinimas naudojant detektorių
3. Jei vertinimo metu nustatoma, jog AE yra pakankamai geros kokybės (pvz. detektorius klasifikuoja kaip nekenksmingą), seka baigiama
4. atliekama selekcija – dažniausiai pasirenkami geriausiai įvertinti populiacijos AE, tačiau galimos ir kitos selekcijos strategijos
5. atliekamas selekcijos atrinktų AE kryžminimas (po 2) taip sukuriant naują AE, turintį po dalį genų iš abiejų kryžmintų AE
6. tam tikrai daliai AE atliekama dalies genų mutacija
7. gaunama nauja populiacijos karta ir seka kartojama nuo 2-o žingsnio

[YPT22]

1.6. Nevalidaus PE formato problema

2. Modelių vertinimas

3. Eksperimentinis tyrimas

Rezultatai ir išvados

Rezultatų ir išvadų dalyje turi būti aiškiai išdėstomi pagrindiniai darbo rezultatai (kažkas išanalizuota, kažkas sukurta, kažkas įdiegta) ir pateikiamos išvados (daromi nagrinėtų problemų sprendimo metodų palyginimai, teikiamos rekomendacijos, akcentuojamos naujovės).

Šaltiniai

[AKF+18] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, P. Roth. *Learning to Evade Static PE Machine Learning Malware Models via Reinforcement Learning*. 2018-01-30. [žiūrėta 2024-09-30]. Prieiga per internetą: https://doi.org/10.48550/arXiv.1801.08917.

[CSD19] R. L. Castro, C. Schmitt, G. Dreo. AIMED: Evolving Malware with Genetic Programming to Evade Detection. Iš: *2019 18th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/13th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE)*. 2019, p. 240–247 [žiūrėta 2024-09-23]. ISSN 2324-9013. Prieiga per internetą: https://doi.org/10.1109/TrustCom/BigDataSE.2019.00040.

[DCB+21] L. Demetrio, S. E. Coull, B. Biggio, G. Lagorio, A. Armando, F. Roli. Adversarial EXEmples: A Survey and Experimental Evaluation of Practical Attacks on Machine Learning for Windows Malware Detection. *ACM Trans. Priv. Secur.* 2021, tomas 24, numeris 4, 27:1–27:31 [žiūrėta 2024-09-30]. ISSN 2471-2566. Prieiga per internetą: https://doi.org/10.1145/3473039.

[FWL+19] Z. Fang, J. Wang, B. Li, S. Wu, Y. Zhou, H. Huang. Evading Anti-Malware Engines With Deep Reinforcement Learning. *IEEE Access*. 2019, tomas 7, p. 48867–48879 [žiūrėta 2024-09-18]. ISSN 2169-3536. Prieiga per internetą: https://doi.org/10.1109/ACCESS.2019.2908033.

[HT17] W. Hu, Y. Tan. *Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN*. 2017-02-20. [žiūrėta 2024-09-18]. Prieiga per internetą: http://arxiv.org/abs/1702.05983.

[YPT22] J. Yuste, E. G. Pardo, J. Tapiador. Optimization of Code Caves in Malware Binaries to Evade Machine Learning Detectors. *Computers & Security*. 2022, tomas 116, p. 102643 [žiūrėta 2024-10-07]. ISSN 0167-4048. Prieiga per internetą: https://doi.org/10.1016/j.cose.2022.102643.

[RSR+18] I. Rosenberg, A. Shabtai, L. Rokach, Y. Elovici. Generic Black-Box End-to-End Attack Against State of the Art API Call Based Malware Classifiers. Iš: M. Bailey, T. Holz, M. Stamatogiannakis, S. Ioannidis (sudarytojai). *Research in Attacks, Intrusions, and Defenses*. Cham: Springer International Publishing, 2018, p. 490–510. ISBN 978-3-030-00470-5. Prieiga per internetą: https://doi.org/10.1007/978-3-030-00470-5_23.

[SB15] J. Saxe, K. Berlin. Deep Neural Network Based Malware Detection Using Two Dimensional Binary Program Features. Iš: *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*. 2015, p. 11–20 [žiūrėta 2024-11-04]. Prieiga per internetą: https://doi.org/10.1109/MALWARE.2015.7413680.

[ZCY+24] F. Zhong, X. Cheng, D. Yu, B. Gong, S. Song, J. Yu. MalFox: Camouflaged Adversarial Malware Example Generation Based on Conv-GANs Against Black-Box Detectors. *IEEE Transactions on Computers*. 2024, tomas 73, numeris 4, p. 980–993 [žiūrėta 2024-09-15]. ISSN 1557-9956. Prieiga per internetą: https://doi.org/10.1109/TC.2023.3236901.

[ZHZ+22] F. Zhong, P. Hu, G. Zhang, H. Li, X. Cheng. Reinforcement Learning Based Adversarial Malware Example Generation against Black-Box Detectors. *Computers & Security*. 2022, tomas 121, p. 102869 [žiūrėta 2024-09-14]. ISSN 0167-4048. Prieiga per internetą: https://doi.org/10.1016/j.cose.2022.102869.

[ZZY+22] E. Zhu, J. Zhang, J. Yan, K. Chen, C. Gao. N-Gram MalGAN: Evading Machine Learning Detection via Feature n-Gram. *Digital Communications and Networks*. 2022, tomas 8, numeris 4, p. 485–491 [žiūrėta 2024-09-23]. ISSN 2352-8648. Prieiga per internetą: https://doi.org/10.1016/j.dcan.2021.11.007.

Sąvokų apibrėžimai

Adversarial attacks. Varžymosi principais pagrįstos atakos. 3, 4

Decision Boundary. Sprendimų priėmimo riba. 3

Karkasas. Nurodo specifines technologijas, naudojamus požymius ir perturbacijas, siekiamus tikslus AE generacijai. Skirtas apibrėžti procesą ir įrankius, kuriuos naudojant būtų galima generuoti nurodytų tikslų siekiančius AE. 2, 4, 5, 7, 8

Maišymo Funkcija. *angl. Hash Function*. Tai funkcija $f : \{0,1\}^* \to \{0,1\}^m$. Naudojama, kai iš begalinės įvesčių erdvės norima gauti fiksuoto dydžio (m) išvestį. 4

Signature. Kodo/programos pėdsakas. 3
Strategija. *angl. Policy*. RL modelio atliekama veiksmų seka. 8
Surogatinis Modelis. ML modelis, aproksimuojantis kitą ML modelį, kurio parametrai (svoriai) nėra žinomi. 4, 7

Zero-sum Game. Dviejų žaidėjų žaidimas, kuriame galimas vienas laimėtojas. Laimėtojo laimėta suma yra lygi pralaimėtojo pralaimėtai sumai. 7

Santrumpos

AE. Varžymosi principais pagrįstomis atakomis obfuskuoti kenkėjiško kodo pavyzdžiai (*angl. Adversarial Examples*)

API. *angl. Application Programming Interface*

DI. Dirbtinis Intelektas

DLL. *angl. Dyanmic-Link Library*

GA. Genetiniais algoritmais pagrįstas ML modelis

GAN. Generatyviniai Priešiški Tinklai (*angl. Generative Adversarial Networks*) (generatyviniai varžymosi principais pagrįsti tinklai)

ML. Mašininis Mokymasis (*angl. Machine Learning*)

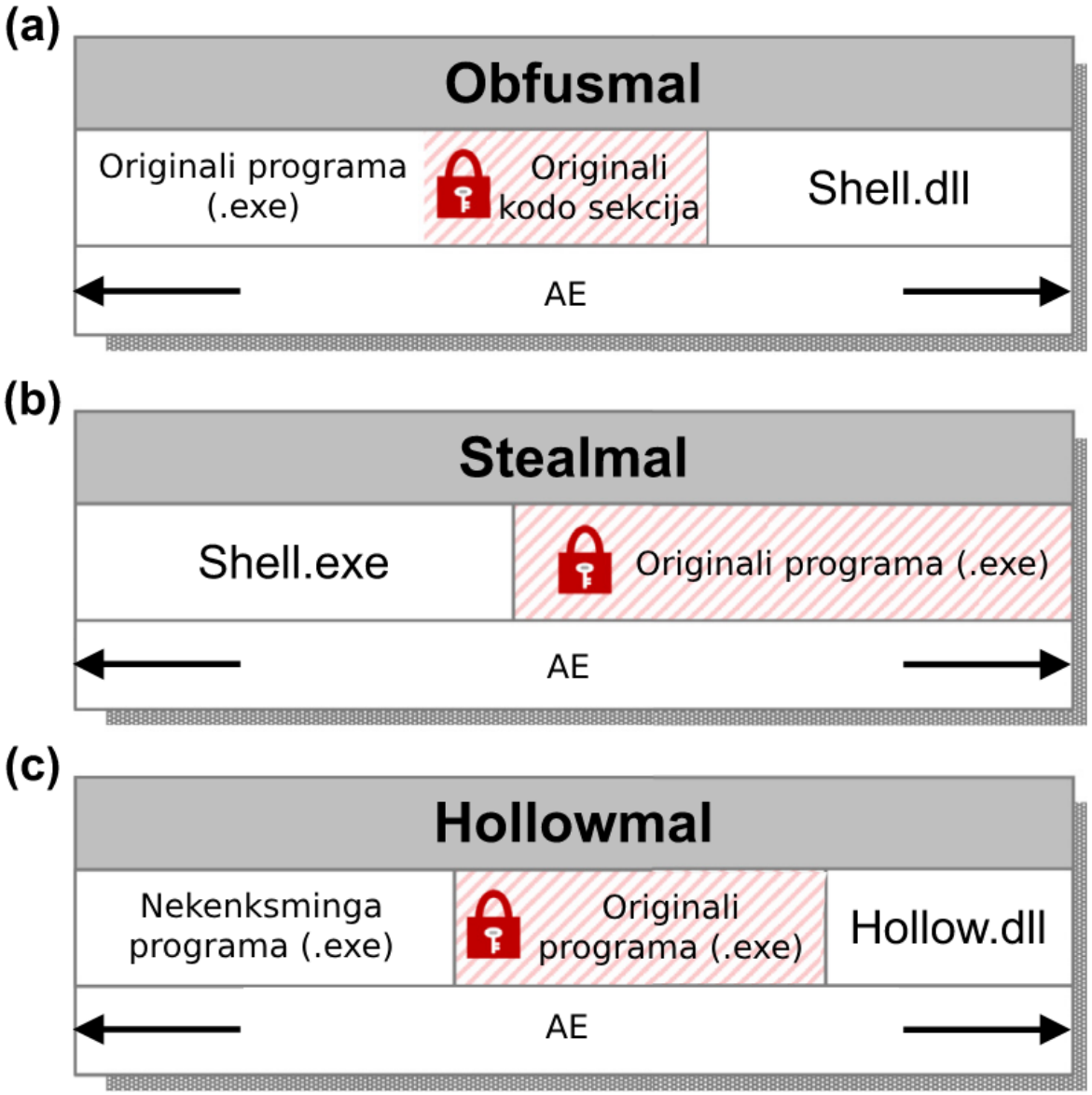
NLP. Skaitmeninis natūraliosios kalbos apdorojimas (*angl. Natural Language Processing*)

PE. *angl. Portable Executable*

RL. Skatinamasis Mokymasis (*angl. Reinforcement Learning*)

Priedai

Priedas nr. 1



1 pav. Obfusmal (a), Stealmal (b) ir Hollowmal (c) perturbacijų veikimo principų iliustracijos [ZHZ+22]