VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

PROGRAMŲ SISTEMŲ STUDIJŲ PROGRAMA

Bakalauro baigiamasis darbas

Varžymosi principais grįstų atakų aptikimas naudojant paaiškinamo dirbtinio intelekto metodą kenkėjiškų programų kontekste

Defense Against Adversarial Malware Obfuscation Attacks Using Explainable Artificial Intelligence

Liudas Kasperavičius

Darbo vadovas : prof. dr. Olga Kurasova

Recenzentas : assoc. prof. Linas Petkevičius

Vilnius
2025

Turinys

Terminų žodynas

Karkasas *(angl. Framework)*. 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, 6, 7, 10, 11, 12, 13, 14, 15

Maišymo funkcija. Tai funkcija $f : \{0,1\}^* \to \{0,1\}^m$. Naudojama, kai iš begalinės įvesčių erdvės norima gauti fiksuoto dydžio (m) išvestį 3, 6

Nulinės sumos žaidimas *(angl. Zero-Sum Game)*. Dviejų žaidėjų žaidimas, kuriame galimas vienas laimėtojas. Laimėtojo laimėta suma yra lygi pralaimėtojo pralaimėtai sumai 10

Pėdsakas. Programos struktūros ir požymių santrauka, beveik unikaliai identifikuojanti programą (pvz., maišymo funkcija) 5

Q-Funkcija *(angl. Q-Function)*. $Q : S \times A \to \mathbb{R}$, čia S – galimų būsenų erdvė *(angl. State Space)*, A – galimų veiksmų erdvė *(angl. Action Space)* 12

Sprendimų priėmimo riba *(angl. Decision Boundary)*. Paprasčiausiems ML modeliams tai yra kreivė plokštumoje. Sudėtingesniems – daugiadimensiniams modeliams – daugdara *(angl. manifold)* 5

Strategija *(angl. Policy)*. Tai funkcija $\pi : S \times A \to \{0,1\}$, čia S – galimų būsenų erdvė *(angl. State Space)*, A – galimų veiksmų erdvė *(angl. Action Space)*. Šią funkciją RL modelis „išmoksta“ mokymosi metu 12

Surogatinis Modelis *(angl. Surrogate Model)*. ML modelis, aproksimuojantis kitą ML modelį, kurio parametrai (svoriai) nėra žinomi 10, 11, 12, 13, 14, 15

Varžymosi principais pagrįstos atakos *(angl. Adversarial Attacks)*. Tai atakos, pritaikytos „apgauti“ ML klasifikatorius 5, 6, 9, 11, 12, 15, 16

Santrumpos

AE. Varžymosi principais pagrįstomis atakomis obfuskuoti kenkėjiško kodo pavyzdžiai *(angl. adversarial examples)* 2, 3, 5, 7, 10, 11, 13, 14, 16

API. *angl. Application Programming Interface* 6, 8, 10, 15

CNN. *angl. convolutional neural network* 11, 12

DI. Dirbtinis intelektas *(angl. artificial intelligence)* 5, 16

DLL. Dinamiškai susieta biblioteka *(angl. dynamic link library)* 6, 8, 9, 11, 15

GA. Genetiniais algoritmais pagrįstas ML modelis *(angl. genetic algorithm)* 14

GAN. Generatyviniai priešiški tinklai *(angl. generative adversarial networks)* 10, 11, 14

GBDT. *angl. gradient boosted decision trees* 11, 12, 14, 15

KNN. *angl.* K*-Nearest Neighbours* 11

ML. Mašininis mokymasis *(angl. machine learning)* 3, 4, 5, 6, 10, 11, 12, 13, 14, 15

NLP. Skaitmeninis natūraliosios kalbos apdorojimas *(angl. natural language processing)* 6

PE. *angl. portable executable* 2, 6, 7, 8, 15

RL. Skatinamasis mokymasis *(angl. reinforcement learning)* 3, 12, 13, 14

SVM. *angl. support vector machine* 10, 11

Įvadas

Pastaraisiais metais kenkėjiškas kodas ir programos kuriamos itin sparčiai ($\sim$450000 kenkėjiškų programų per dieną 2024 m. AV-TEST[1] duomenimis). Kenkėjiško kodo aptikimo programos, kurios tradiciškai remiasi programų pėdsakais, nespėja atnaujinti pėdsakų duomenų bazių pakankamai greitai. Dėl to 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 [CSD19; HT17; RSR+18; ZHZ+22]. Šių atakų principas yra ML modelio – klasifikatoriaus – sprendimų priėmimo ribos 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 [AKF+18]. Todėl šiame darbe nagrinėjami modeliai, gebantys generuoti varžymosi principais pagrįstų atakų obfuskuotus kenkėjiško kodo pavyzdžius (AE) „juodos dėžės“ atvejams.

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

Uždaviniai:

1. Apžvelgti kenkėjiško kodo obfuskacijos metodus.
2. Nustatyti kriterijus varžymosi principais grįstų atakų karkasams ir juos įvertinti.
3. Atlikti eksperimentinį tyrimą su vienu iš įvertintų karkasų ir patikrinti vertinimo rezultatus.

[1] https://www.av-test.org/en/statistics/malware

1. Literatūros apžvalga

1.1. Naudojami kenkėjiškų programų požymiai

Varžymosi principais pagrįsta ataka 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 ERRORkarkasas apibrėžimuose, priklausomai nuo jų specifikos ir tikslų, neretai pateikiami jų vertinami programų požymiai. Šiame poskyryje išskiriami ir klasifikuojami mokslinėje literatūroje minimi požymiai.

1.1.1. PE formato programų požymiai

Išskiriami šie pagrindiniai PE formato programų požymiai:

- **DLL vardai (arba API vardai [HT17])** [ZCY+24]. PE faile turi būti nurodyti visi naudojami DLL ir jų API. Prieš pradedant mokyti ML modelį, atliekama visų turimų programų analizė ir nustatoma visų naudojamų DLL ar jų API 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

Baitų lygio požymiai gali būti ištraukiami iš bet kokio formato failų. Mokslinėje literatūroje minimi šie pagrindiniai baitų lygio požymiai:

- **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 n 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) [HT17]. Perturbacijų parinkimas įeina į karkaso apibrėžimą. Šiame poskyryje aptariamos mokslinėje literatūroje minimos perturbacijos.

1.2.1. Baitų lygio perturbacijos

Pačias paprasčiausias baitų lygio perturbacijas galima taikyti bet kokio formato failams, tačiau labiau prasmingos perturbacijos taikomos PE formato failams. Išskiriamos šios pagrindinės 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[2]. 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[2] S per n, atsiranda n baitų laisvos vietos iki sekcijos pradžios, kurią galima keisti be jokios įtakos programos veikimui.

1.2.2. Semantinės perturbacijos

Semantinių perturbacijų įgyvendinimas taip pat atliekamas baitų lygyje, tačiau šie pokyčiai turi aukštesnio lygio prasmę. Išskiriamos šios 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 $A - (-B)$.

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

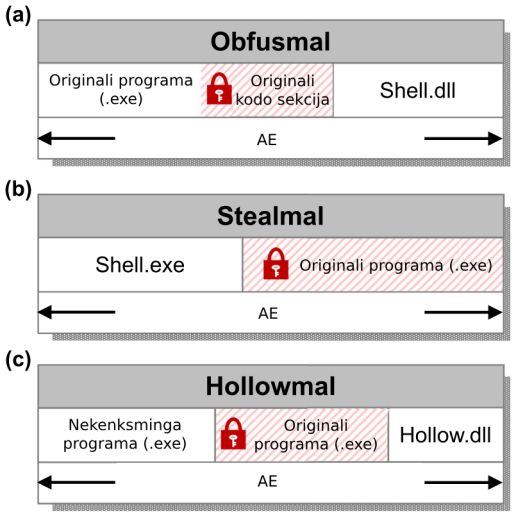
1.2.3. Kompleksinės perturbacijos

Kompleksinės perturbacijos yra pritaikomos tam tikriems tikslams. Obfuskacijos ir varžymosi principais pagrįstų atakų tikslams literatūroje minimos šios 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 pav. *Obfusmal (a), Stealmal (b) ir Hollowmal (c) perturbacijų veikimo principų iliustracijos. Adaptuota iš [ZHZ+22]*

1.3. GAN tipo modelių karkasai

GAN modelių karkasai paremti generatyviniais priešiškais tinklais (GAN), kurių veikimo principas yra du neuroniniai tinklai (generatorius ir diskriminatorius), žaidžiantys nulinės sumos žaidimas [CDH+16]. Kenkėjiško kodo obfuskacijos kontekste ir ypač „juodos dėžės“ atvejais, diskriminatorius atlieka surogatinis modelis vaidmenį. Bendras GAN modelių mokymosi etapas yra tokia seka [HT17; ZCY+24; ZZY+22]:

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 klasifikuoja sugeneruotą AE (kenkėjiškas / nekenkėjiškas). Pagal klasifikacijos rezultatą skaičiuojamos diskriminatoriaus ir generatoriaus nuostolių funkcijos reikšmės (diskriminatoriaus nuostolių funkcijos reikšmė priklauso nuo tikro detektoriaus klasifikacijos).
4. Visa seka kartojama nustatytą kiekį kartų.

1.3.1. *MalGAN*

Tai vienas iš pirmųjų ir populiariausių GAN tipo modelių ERRORkarkasas. „Juodos dėžės“ detektoriai čia apibrėžiami kaip populiarūs ML klasifikatoriai, tokie, kaip MLP *(angl. Multilayer Perceptron)*, RF *(angl. Random Forest)*, DT *(angl. Decision Tree)*, SVM *(angl. Support Vector Machine)*. *MalGAN* karkaso [HT17] tikslas ir apibrėžimas pateikiami 1-oje lentelėje.

1 lentelė. *MalGAN karkasas*

Tikslas	Efektyviai išvengti AE aptikimo, kai ML kenkėjiškų programų detektoriaus įgyvendinimas nežinomas („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 [HT17] 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[3].
Perturbacijos	Semantinės perturbacijos (1.2.2.) – nereikalingų API vardų požymių pridėjimas.

[3] autoriai nagrinėja „juodos dėžės“ atvejį su prielaida, jog detektoriaus naudojami požymiai yra žinomi.

1.3.2. *N-gram MalGAN*

Šis karkasas remiasi *MalGAN* (1.3.1.) karkasu ir siekia jį pagerinti. *N-gram MalGAN* karkaso [ZZY+22] tikslas ir apibrėžimas pateikiami 2-oje lentelėje.

2 lentelė. *N-gram MalGAN karkasas*

Tikslas	Supaprastinti, pagreitinti ir pagerinti varžymosi principais pagrįstas atakas. Pašalinti prielaidas[3] apie detektorių „juodos dėžės“ atvejais.
Surogatinis modelis	Surogatinio modelio veikimas ir architektūra tokia pati, kaip ir *MalGAN* (1.3.1.).
ML modelis	Pagrindinio modelio veikimas ir architektūra labai panašūs į *MalGAN* (1.3.1.), tačiau norėdami stabilizuoti mokymosi procesą, autoriai siūlo nenaudoti „triukšmo“ vektoriaus. Vietoje to, generatoriaus išvestis (n-matis vektorius) modifikuojama nekeičiant pirmų m dimensijų, o kitas $n - m$ pakeičiant nekenksmingų programų požymiais.
Požymiai	Baitų lygio požymiai (1.1.2.) – n-gramos.
Perturbacijos	Autoriai neatliko eksperimentų su perturbuotomis programomis, tačiau pažymi, jog norint gauti sugeneruotus požymių vektorius užtenka pridėti reikiamus baitus programos gale. Tai atitinka 1.2.1. apibrėžtą baitų lygio perturbaciją *ARBE*, tik šiuo atveju pridedami baitai nebūtų atsitiktiniai, o norimos n-gramos.

1.3.3. *MalFox*

MalFox taip pat remiasi *MalGAN* (1.3.1.), tačiau siekia kurti AE realiomis sąlygomis, dėl to atlieka esminius pakeitimus. *MalFox* karkaso [ZCY+24] tikslas ir apibrėžimas pateikiami 3-oje lentelėje.

3 lentelė. *MalFox karkasas*

Tikslas	Generuoti AE, kurių neaptiktų komerciniai detektoriai (prieš tai aptarti karkasai eksperimentams kaip nepriklausomą detektorių naudojo tokius ML modelius, kaip SVM, KNN, GBDT ir kt., bet ne komercinius detektorius). Šio karkaso detektorius yra *VirusTotal* (viešai prieinama paslauga, agreguojanti virš 70 komercinių kenkėjiškų programų detektorių).
Surogatinis modelis	Surogatinis modelis, kaip ir kituose GAN tipo modelių karkasuose, naudojamas kaip diskriminatorius. Įvestis – perturbuota programa. Išvestis – klasifikacija į kenksmingą arba nekenksmingą. Įgyvendinimas – konvoliucinis neuroninis tinklas (CNN).
ML modelis	Standartinis GAN generatorius, požymių vektorių sujungiantis su „triukšmo“ vektoriumi. Įgyvendinimas – konvoliucinis neuroninis tinklas (CNN).
Požymiai	PE formato programų požymiai (1.1.1.) – DLL vardai.

Perturbacijos	Visos kompleksinės perturbacijos (1.2.3.).

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. Šiuo atveju aplinkos būsenų erdvė S yra požymių vektorių erdvė. Agentas – tai algoritmas ar neuroninis tinklas, kurio tikslas yra surasti optimalią strategiją strategija *(angl. policy)* ko. Šiuo atveju strategijos veiksmų erdvė A susideda iš perturbacijų (žr. 1.2.) [FWL+19]. Bendras RL modelių mokymosi etapas yra tokia seka [FWL+19; ZHZ+22]:

1. Agentas, naudodamas dabartinę aplinkos būseną ir praeito veiksmo atlygį *(angl. reward)*, parenka sekantį veiksmą iš galimų veiksmų aibės ir taiko mokymosi algoritmą (algoritmas priklauso nuo agento įgyvendinimo).

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.

1.4.1. *DQEAF*

Šis karkasas taiko gilųjį skatinamąjį mokymąsi, kai agentas implementuojamas kaip gilusis neuroninis tinklas. *DQEAF* karkaso [FWL+19] tikslas ir apibrėžimas pateikiami 4-oje lentelėje.

4 lentelė. *DQEAF karkasas*

Tikslas	Parodyti, jog ML kenkėjiškų programų detektoriai, ypač modeliai, išmokyti prižiūrimu mokymusi, yra pažeidžiami varžymosi principais pagrįstoms atakoms.
Surogatinis modelis	RL karkasuose nenaudojami surogatiniai modeliai. Kaip „juodos dėžės“ detektorius pasirinktas GBDT modelis.
ML modelis	Agentas implementuotas kaip gilusis Q-tinklas (CNN praplėtimas, kai tinklas naudojamas kaip Q-funkcijos aproksimacija). Taip pat taikomas prioritetizuotas patirčių pakartojimo metodas *(angl. prioritized experience replay)*, kuomet agentas mokomas tik su aukštą atlygį gavusiais perėjimais ($S \times A$).

Požymiai	Požymių vektorius taip pat apibrėžia visų būsenų erdvę S. Šiuo atveju $S = \mathbb{R}^{513}$. Baitų lygio požymiai (1.1.2.) – baitų/entropijos histograma.
Perturbacijos	Perturbacijos apibrėžia visų galimų agento veiksmų erdvę A. Šiuo atveju $A = \{0,1\}^4$. Baitų lygio perturbacijos (1.2.1.): • *ARBE* • *ARI* • *ARS* • *RS*

1.4.2. *MalInfo*

MalInfo remiasi *MalFox* (1.3.3.). *MalInfo* karkaso [ZHZ+22] tikslas ir apibrėžimas pateikiami 5-oje lentelėje.

5 lentelė. *MalInfo karkasas*

Tikslas	Surasti optimalią obfuskacijos strategiją konkrečiai programai, pagal kurią sukurtas AE nebūtų aptiktas komercinių kenkėjiškų programų detektorių.
Surogatinis modelis	RL surogatinis modelis nenaudojamas. „Juodos dėžės“ detektoriumi pasirinkti komerciniai detektoriai (*VirusTotal*).
ML modelis	Agentas implementuotas kaip klasikiniai ML algoritmai (konkrečiai dinaminis programavimas ir skirtumų laike *(angl. temporal difference)* algoritmas).
Požymiai	Agentas nėra neuroninis tinklas ir požymių iš programos netraukia. Agentas mokosi tik iš perėjimų, o būsenų erdvė S yra originali programa ir perturbuoti jos variantai. Teoriškai perturbuotų programos variantų galėtų būti be galo daug, tuomet $S = A^\infty, \lvert S\rvert = \aleph_0$, tačiau autoriai nurodo, jog daugiau nei 3 sluoksniai kompleksinių perturbacijų reikšmingai paveikia programos veikimo laiką, o tai gali „sukelti įtarimų“ komerciniams detektoriams. Todėl pasirinkta $S = A^3$.
Perturbacijos	$A = \{0,1,2,3\}$ • *Null* perturbacija – naudinga tik formaliam pilnumui (atitinka nulinį A veiksmą). • Visos kompleksinės perturbacijos (1.2.3.), t. y. tokios pačios, kaip ir *MalFox* (1.3.3.) karkaso.

1.5. Genetinių algoritmų tipo modelių karkasai

Genetinai algoritmai (GA) yra viena seniausių mašininio mokymosi ML apraiškų; jų veikimas paremtas evoliucija [CSD19]. Kenkėjiškų programų obfuskacijai AE generavimas taikant GA yra tokia seka [YPT22]:

1. Sukuriama pradinė populiacija (perturbacijos metodai pradinei populiacijai priklauso nuo karkaso).
2. Atliekamas tinkamumo *(angl. fitness)* vertinimas.
3. Atliekama selekcija – dažniausiai pasirenkami geriausiai įvertinti populiacijos AE, tačiau galimos ir kitos selekcijos strategijos.
4. Atliekamas selekcijos atrinktų AE kryžminimas (po 2) taip sukuriant naują AE, turintį po dalį genų iš abiejų kryžmintų AE.
5. Tam tikrai daliai AE atliekama dalies genų mutacija.
6. Vertinama, ar sugeneruoti AE atitinka kriterijus (vertina detektorius).
7. Jei kriterijai nėra tenkinami, seka kartojama nuo 2-o žingsnio.

1.5.1. *AIMED*

AIMED karkaso [CSD19] tikslas ir apibrėžimas pateikiami 6-oje lentelėje.

6 lentelė. *AIMED karkasas*

Tikslas	AE generavimo greičio padidinimas ir modelių kompleksiškumo sumažinimas, lyginant su GAN ir RL tipo modelių karkasais.
Surogatinis modelis	Surogatinis modelis nenaudojamas. Naudojami „juodos dėžės“ detektoriai yra **3 komerciniai** (*Kaspersky, ESET, Sophos*) ir vienas ML modelis – GBDT.
ML modelis	Klasikinis GA modelis – veikimas visiškai atitinką bendrą seką. Tinkamumo *(angl. fitness)* vertinamas remiasi AE požymių vektoriaus panašumu į originalios programos požymių vektorių (kuo mažiau panašūs, tuo tinkamumo įvertinimas didesnis).
Požymiai	Baitų lygio požymiai (1.1.2.) – atskiras n-gramų atvejis, kai $n = 1$.
Perturbacijos	Baitų lygio perturbacijos[4] (1.2.1.).

[4] autoriai rėmėsi perturbacijomis, aprašytomis [AKF+18]

1.5.2. *GAMMA*

GAMMA karkaso [DBL+21] tikslas ir apibrėžimas pateikiami 7-oje lentelėje.

7 lentelė. *GAMMA karkasas*

Tikslas	Efektyvus (neaptikimo šansų didinimas naudojant perturbacijas, paremtas nekenksmingomis programomis) varžymosi principais pagrįstų atakų kūrimas.
Surogatinis modelis	Surogatinis modelis nenaudojamas. GBDT ir *MalConv* pasirinkti kaip „juodos dėžės“ detektoriai.
ML modelis	Pagrindinė modelio idėja yra požymių ištraukimas iš nekenksmingų programų ir jų pridėjimas, naudojant tam pritaikytas perturbacijas, į kenksmingas programas kiekvienos populiacijos generavimo metu. Tinkamumo *(angl. fitness)* ir kriterijų vertinimas atliekamas naudojant detektorių ir pridėtų požymių dydį baitais (norima pridėti kuo mažiau požymių).
Požymiai	• PE formato programų požymiai (1.1.1.). • Kodas sekcijose (nestandartinis požymis).
Perturbacijos	• Visos baitų lygio perturbacijos (1.2.1.), gebančios pridėti baitus. • Autoriai pažymi, jog gali būti naudojama ir DLL / API vardų pridėjimo semantinė perturbacija (1.2.2.).

1.6. Nevalidaus PE formato problema

Anderson et al. [AKF+18], atlikdami eksperimentus su funkcionalumą išlaikančiomis perturbacijomis PE formato failams, pastebėjo, jog ne visais atvejais perturbuotos programos veikia teisingai. Dėl *Windows* operacinės sistemos PE formato failų interpretavimo ir paleidimo specifikos, programas įmanoma parašyti tokiu būdu, jog pakeitus kodo ar kitų sekcijų turinį nekeičiant originalių mašininio kodo instrukcijų, programa neveiktų. Techniškai, programų rašymas tokiu būdu pažeidžia patį PE formato standartą, tačiau šią praktiką neretai naudoja kenkėjiškų programų autoriai.

Norint visiškai išvengti nevalidaus PE formato problemos tenka taikyti perturbacijas, nekeičiančias originalių programų, o taikančias kitokius obfuskacijos metodus. Iš 1.2. poskyryje aptartų perturbacijų, tokias sąlygas atitinka tik 2 kompleksinės perturbacijos (1.2.3.) – *Stealmal* ir *Hollowmal*.

1.7. AE perkeliamumas

Perkeliamumas DI modeliuose dažniausiai suprantamas kaip žinių perkeliamumas *(angl. knowledge transferability)*. Tačiau tiriant varžymosi principais pagrįstas atakas buvo pastebėtas ir AE perkeliamumas *(angl. adversarial sample transferability)*. Tai reiškia, jog gebant sukurti AE, kuriuos neteisingai klasifikuoja vienas modelis, tikėtina, jog kitas (tos pačios paskirties) modelis taip pat klasifikuos AE neteisingai. Be to, nustatyta, jog AE perkeliamumo savybė galioja skirtingų architektūrų modeliams [DCB+21]. Šia savybe remiasi visos „juodos dėžės“ atvejams pritaikytos atakos.

Literatūra ir 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. https://doi.org/10.48550/arXiv.1801.08917. (Žiūrėta 2024-09-30).

[CDH+16] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, P. Abbeel. *InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets*. 2016. https://doi.org/10.48550/arXiv.1606.03657. (Žiūrėta 2024-10-07).

[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, puslapiai 240–247. https://doi.org/10.1109/TrustCom/BigDataSE.2019.00040. (Žiūrėta 2024-09-23).

[DBL+21] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, A. Armando. „Functionality-Preserving Black-Box Optimization of Adversarial Windows Malware“. Iš: *IEEE Transactions on Information Forensics and Security* 16 (2021), puslapiai 3469–3478. ISSN: 1556-6021. https://doi.org/10.1109/TIFS.2021.3082330. (Žiūrėta 2024-10-14).

[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“. Iš: *ACM Trans. Priv. Secur.* 24.4 (2021), 27:1–27:31. ISSN: 2471-2566. https://doi.org/10.1145/3473039. (Žiūrėta 2024-09-30).

[FWL+19] Z. Fang, J. Wang, B. Li, S. Wu, Y. Zhou, H. Huang. „Evading Anti-Malware Engines With Deep Reinforcement Learning“. Iš: *IEEE Access* 7 (2019), puslapiai 48867–48879. ISSN: 2169-3536. https://doi.org/10.1109/ACCESS.2019.2908033. (Žiūrėta 2024-09-18).

[HT17] W. Hu, Y. Tan. *Generating Adversarial Malware Examples for Black-Box Attacks Based on GAN*. 2017. (Žiūrėta 2024-09-18).

[YPT22] J. Yuste, E. G. Pardo, J. Tapiador. „Optimization of Code Caves in Malware Binaries to Evade Machine Learning Detectors“. Iš: *Computers & Security* 116 (2022), puslapis 102643. ISSN: 0167-4048. https://doi.org/10.1016/j.cose.2022.102643. (Žiūrėta 2024-10-07).

[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: *Research in Attacks, Intrusions, and Defenses*. Sudarė M. Bailey, T. Holz, M. Stamatogiannakis, S. Ioannidis. Cham: Springer International Publishing, 2018, puslapiai 490–510. ISBN: 978-3-030-00470-5. 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, puslapiai 11–20. https://doi.org/10.1109/MALWARE.2015.7413680. (Žiūrėta 2024-11-04).

[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“. Iš: *IEEE Transactions on Computers* 73.4 (2024), puslapiai 980–993. ISSN: 1557-9956. https://doi.org/10.1109/TC.2023.3236901. (Žiūrėta 2024-09-15).

[ZHZ+22] F. Zhong, P. Hu, G. Zhang, H. Li, X. Cheng. „Reinforcement Learning Based Adversarial Malware Example Generation against Black-Box Detectors“. Iš: *Computers & Security* 121 (2022), puslapis 102869. ISSN: 0167-4048. https://doi.org/10.1016/j.cose.2022.102869. (Žiūrėta 2024-09-14).

[ZZY+22] E. Zhu, J. Zhang, J. Yan, K. Chen, C. Gao. „N-Gram MalGAN: Evading Machine Learning Detection via Feature n-Gram“. Iš: *Digital Communications and Networks* 8.4 (2022), puslapiai 485–491. ISSN: 2352-8648. https://doi.org/10.1016/j.dcan.2021.11.007. (Žiūrėta 2024-09-23).