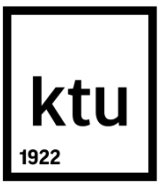


KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

Donatas Kvietkus

PROGRAMINĖS ĮRANGOS APSAUGA NUO NELEGALAUS NAUDOJIMO KEIČIANT JOS FUNKCIONALUMĄ

Baigiamasis magistro darbas

Vadovas
Doc. dr. Jonas Čeponis

KAUNAS, 2015

KAUNO TECHNOLOGIJOS UNIVERSITETAS

INFORMATIKOS FAKULTETAS

KOMPIUTERIŲ KATEDRA

TVIRTINU
Katedros vedėjas
(parašas) Prof. dr. Algimantas Venčkauskas
(data)

PROGRAMINĖS ĮRANGOS APSAUGA NUO NELEGALAUS NAUDOJIMO KEIČIANT JOS FUNKCIONALUMĄ

Baigiamasis magistro darbas

Informacijos ir informacinių technologijų sauga (kodas 621E10003)

Vadovas
(parašas) Doc. dr. Jonas Čeponis
(data)

Recenzentas
(parašas) Lekt. Ramūnas Kubiliūnas
(data)

Projektą atliko
(parašas) Donatas Kvietkus
(data)

KAUNAS, 2015

KAUNO TECHNOLOGIJOS UNIVERSITETAS

Informatikos fakultetas

(Fakultetas)

Donatas Kvietkus

(Studento vardas, pavardė)

Informacijos ir informacinių technologijų sauga (kodas 621E10003)

(Studijų programos pavadinimas, kodas)

Baigiamojo projekto „Programinės įrangos apsauga nuo nelegalaus naudojimo keičiant jos funkcionalumą“

AKADEMINIO SĄŽININGUMO DEKLARACIJA

20 15 m. gegužės 25 d.

Kaunas

Patvirtinu, kad mano **Donato Kvietkaus** baigiamasis projektas tema „Programinės įrangos apsauga nuo nelegalaus naudojimo keičiant jos funkcionalumą“ yra parašytas visiškai savarankiškai, o visi pateikti duomenys ar tyrimų rezultatai yra teisingi ir gauti sąžiningai. Šiame darbe nei viena dalis nėra plagijuota nuo jokių spausdintinių ar internetinių šaltinių, visos kitų šaltinių tiesioginės ir netiesioginės citatos nurodytos literatūros nuorodose. Įstatymų nenumatytų piniginių sumų už šį darbą niekam nesu mokėjęs.

Aš suprantu, kad išaiškėjus nesąžiningumo faktui, man bus taikomos nuobaudos, remiantis Kauno technologijos universitete galiojančia tvarka.

(vardą ir pavardę įrašyti ranka)	*(parašas)*

Kvietkus D. Programinės įrangos apsauga nuo nelegalaus naudojimo keičiant jos funkcionalumą. Magistro baigiamasis projektas / vadovas doc. dr. Jonas Čeponis; Kauno technologijos universitetas, Informatikos fakultetas, Kompiuterių katedra.

Kaunas, 2015. 44 psl.

SANTRAUKA

Magistro baigiamajame darbe yra analizuojami programinės įrangos saugojimo metodai ir technologiniai sprendimai, kurių dėka gali būti sukuriamas kelių lygių apsaugos modelis. Formaliam sudaryto apsaugos modelio veikimo principų aprašui panaudotos veiklos diagramos. Aprašyti apsaugos variantai, skirti laisvam programuotojo pasirinkimui. Esminis sudaryto apsaugos modelio principas yra sukurti iliuzinę programinės įrangos apsaugą. Tikrasis apsaugos mechanizmas suveiks tik programos veikimo metu. Sukurtas apsaugos mechanizmas buvo panaudotas programinėje įrangoje ir patikrintas jo veikimas. Įveikus pirmos dalies apsaugą, programinė įranga gali būti naudojama pilnu funkcionalumu, tam, kad susidarytų iliuzinis vaizdas, jog programa yra jau pasisavinta programišiaus, o tikrieji tikrinimo metodai paslėpti kažkur programoje.

Kvietkus D. Software protection against illegal use by changing its functionality. Master thesis / supervisor Assoc. Prof. Dr. Jonas Čeponis; Department of Computer Science, Faculty of Informatics, Kaunas University of Technology

Kaunas, 2015. - 44 p.

SUMMARY

In Masters final work are being analyzed software protection methods and technological solutions, because of which there can be created a few level protection model. For formal created protection model's working principles description were used activity graphs. Described protection choices, is for free programmer's choosing. Essential created protection model's principle is to create illusionary software's protection. The real protection mechanism only activates when the program is being on. Created protection mechanism was used on software to test its working. By overcoming first part protection, software can be used with full functionality, to make illusionary image, that program is already taken over by programmer, when the real checking methods would be hidden somewhere in the program.

Turinys

Lentelių sąrašas 7
Paveikslų sąrašas 8
Terminų ir santrumpų žodynas 9
ĮVADAS 10
1. Programinio kodo apsauga 11
1.1. Programinio kodo maskavimas 11
1.1.1. Apgrąžos inžinerija 13
1.1.2. Vandens ženklai 14
1.2. Programinės įrangos licencijavimas 15
1.2.1. Šifravimo metodai 16
1.3. Gijos 19
1.4. Programiniai įrankiai 19
1.4.1. Operacinė sistema 19
1.4.2. Programavimo kalba 20
1.4.3. Maskavimo įrankiai 20
1.5. Analizės išvados 22
2. Programinio kodo apsaugos pritaikymas 23
2.1. Programinio kodo skaitymo apsunkinimas 24
2.2. Licencijos failo įterpimas 24
2.3. „Nematomo“ komponento panaudojimas 26
2.4. Laiko žymės tikrinimo modelis panaudojant gijas 27
2.5. Nenaudingo kodo ir paspaudimo kombinacijų panaudojimas 29
3. Apsaugos metodo taikymas programinėje įrangoje 31
3.1. Licencinio failo sukūrimas ir panaudojimas 31
3.2. „Nematomo“ komponento sukūrimas ir panaudojimas 34
3.3. Laiko intervalo fiksavimo ir nuskaitymo realizavimas 35
3.4. Nenaudingo kodo įterpimas 37
3.5. Kodo maskavimas panaudojant jau sukurtus maskavimo įrankius 38
3.6. Programos veikimas 39
4. Išvados 41
5. Literatūra 42

Lentelių sąrašas

1.1. Lentelė Populiariausiai simetrinio šifravimo algoritmai 17
1.2. Lentelė Personalinių kompiuterių operacinių sistemų palyginimas 20
1.3. Lentelė Labiausiai naudojamos programavimo kalbos 20
1.4. Lentelė Kodo maskavimo įrankių palyginimas 21

Paveikslų sąrašas

1.1. Pav. Originalus kodas 12
1.2. Pav. Užmaskuotas kodas 12
1.3. Pav. Asemblerio kalba 13
1.4. Pav. Vandens ženklų taksonomijos medis (Jasvir Nagra, 2001) 14
1.5. Pav. Statinis ir dinaminis vandens ženklas 15
1.6. Pav. Procesas tarp programuotojo ir vartotojo 16
1.7. Pav. Simetrinis šifravimas 17
1.8. Pav. Asimetrinis šifravimas 18
2.1. Pav. Apsaugos metodo schema 23
2.2. Pav. Licencijos failo pavyzdys 25
2.3. Pav. Licencijos tikrinimas 25
2.4. Pav. Atsitiktinai generuojamas komponentas 26
2.5. Pav. Paslėpto mygtuko tikrinimo metodas 27
2.6. Pav. Matomas ir nematomas failas 28
2.7. Pav. Gijų veiklos diagrama 29
2.8. Pav. Nenaudingo kodo sekų diagrama 30
3.1. Pav. Prekybos valdymo sistema 31
3.2. Pav. Licencijos failas prieš šifravimą 32
3.3. Pav. Bitų skaičiavimo algoritmas 32
3.4. Pav. Šifravimo ir atšifravimo kodo fragmentas 33
3.5. Pav. Šifruotas licencijos failas 33
3.6. Pav. Atsitiktinės pozicijos komponento kūrimas 34
3.7. Pav. Failo nuskaitymas įkeliant į atmintį 35
3.8. Pav. Mažos reikšmės failas 36
3.9. Pav. Paslėpto failo kūrimas 36
3.10. Pav. Datų vientisumo tikrinimas 37
3.11. Pav. Nenaudingas kodas 38
3.12. Pav. "ConfuserEx" maskavimo įrankis 39
3.13. Pav. Pasibaigusios licencijos rezultatas 39
3.14. Pav. Modifikuoto licencinio failo rezultatas 40

Terminų ir santrumpų žodynas

Pirminis programos tekstas (angl., *Source code*) - programos tekstas, kurį rašė programuotojas tam tikra programavimo kalba.

Maskavimas (angl., *Obfuscation*) - programinio kodo perdarymas į žmonėms sunkiai suprantamą kodą, skirtas paslėpti pirminį programos tekstą.

Apgrąžos inžinerija (angl., *Reverse code engineering*) - Objekto analizės būdas nustatyti, iš ko ir kaip tas objektas padarytas, pavyzdžiui, iš programos atkuriamas jos algoritmas.

Programišius (angl., *Hacker*) - Asmuo turintis pomėgį programuoti, nagrinėti operacinių sistemų ir kitų programų pirminius tekstus, juos tobulinti.

Kenkėjiška programinė įranga (angl., *Malware*) - programinė įranga, sukurta kompiuterio veikimui trikdyti arba kompiuterį panaudoti kenkimo tikslams.

Bitų sumos tikrinimas (angl., *Checksum*) - sumuojama kiek bitų sudaro tam tikri duomenys.

Įvykis (angl., *Event*) – tam tikro veiksmo atlikimas atsitikus įvykiui (duomenų įvedimas, laiko pasikeitimas ir t.t.).

Vandens ženklai (angl. *Watermarking*) - unikalus žymėjimas tam tikroje kodo dalyje.

Asembleris (angl., *Assembly*) – žemo lygio programavimo kalba.

Dekompiliatorius (angl., *Decompiler*) - apgrąžos funkciją atliekantis įrankis, dažniausiai skirtas dvejetainį kodą paversti į pirminį, aukštesnio lygio programavimo kalbos kodą.

Daugiagijiškumas (angl., *Multithreading*) - procesas, kuriame naudojama daugiau nei viena gija.

IDC (*International Data Corporation*).

NTFS (*New Technology File System***) -** Microsoft sukurta failų sistema, kuri turėtų padidinti duomenų greitaveiką, patikimumą, pagerinti duomenų panaudojimą diske, pridėtas apsaugos valdymas.

MD5 (*Message Digest Algorithm 5*) - maišos funkcija paverčianti tekstą į neskaitomą simbolių aibę pagal algoritmą.

ĮVADAS

Darbas priklauso Informacijos ir informacinių technologijų saugos studijų programai ir specializacijai.

Programinės įrangos apsaugojimas nuo nelegalaus naudojimo yra skausmingas uždavinys kiekvienam programinės įrangos kūrėjui. Dokumentuotų apsaugos metodų yra mažai ir jie visi nėra atsparūs apgrąžos inžinerijai. Šiame darbe bus išanalizuoti viešai aprašomi saugos metodai, ir pateiktas modelis, kuriame tie saugos metodai bus panaudoti kiek kitokiam tikslui. Šis darbas skirtas žmonėms, kurie nori sustiprinti programinės įrangos apsaugą nuo nelegalaus naudojimo.

Darbo tikslas ir uždaviniai

Tikslas

Patobulinti programinės įrangos apsaugos modelį nuo nelegalaus jos panaudojimo įvertinant jau esamus apsaugos metodus.

Uždaviniai

- Išanalizuoti esamus programinės įrangos saugos metodus.
- Sukurti efektyvų programinės įrangos saugos modelio projektą.
- Panaudoti efektyvų programinės įrangos saugos modelį jau egzistuojančioje programoje.
- Patikrinti panaudoto efektyvaus programinės įrangos saugos modelio veikimą.

Darbo rezultatai ir jų svarba

Darbe buvo suprojektuotas ir panaudotas saugos modelis panaudojant esamus saugos metodus, gauti rezultatai įrodo kodėl kūrėjai nedokumentuoja saugos mechanizmų. Žinant programinės įrangos tikrinimo vietas, galima nesudėtingai jas pasikeisti pasinaudojant tam skirtais programiniais įrankiais.

Darbo struktūra

Baigiamąjį magistro darbą sudaro 4 skyriai:

1. Programinio kodo apsauga – išanalizuojami naudojami saugos metodai apsaugoti programinę įrangą, taip pat trumpai aprašomos programinės technologijos, kurios bus naudojamos.
2. Programinio kodo apsaugos pritaikymas – pateikiamas apsaugos modelis, kuris sudarytas iš jau esamų saugos metodų, kurie bus naudojami fiktyviai apsaugai, o pagrindinė apsauga bus realizuojama įprastais metodais.
3. Apsaugos metodo taikymas programinėje įrangoje – išsamiai aprašomi žingsniai, kuriuos reikia atlikti, norint realizuoti antrajame skyriuje aprašytą saugos modelį.
4. Išvados – pateikiamos apibendrintos darbo išvados ir gauti rezultatai.

1. PROGRAMINIO KODO APSAUGA

Programinis kodas yra tam tikrų procedūrų, metodų ir taisyklių visuma, kuri gali vadintis programinė įranga ar jos dalis. Programinė įranga turi vieną ir daugiau kūrėjų, o tai ne kas kitą, kaip intelektinė nuosavybė, į kurią vis kėsinasi programišiai norėdami, tai pasisavinti ar iš to pasipelnyti. Tam jog apsaugoti programinį kodą ar įrangą nuo nelegalaus panaudojimo, kūrėjai neretai bando taikyti įvairius apsaugos metodus, tam jog būtų apsunkintas kelias į piratavimą. Dėl nelegalaus programinės įrangos naudojimo nukenčia netik patys kūrėjai, tačiau ir šalies ekonomika. Per 2013uosius metus nelegaliai programinės įrangos buvo naudojama 43%, o tai sudaro 62,7mlrd JAV dolerių (Alliance, 2015), ir, tai nėra galutinė suma, nes įmonės išleidžia saugumui dar didesnes sumas. Dėl piratinės programinės įrangos pagal IDC prognozes 2014 metais įmonėms teko išleisti 491mlrd, o 2013 metais – 114mlrd JAV dolerių tam, kad atstatytų programinę įrangą, kuri buvo perimta naudoti kenkėjiškais tikslais (Hernandez, 2015).

Didžiausia problema yra tai, jog nėra apsaugos, kuriuos nebūtų įmanoma apeiti, viskas priklauso tik nuo laiko, kada programinis kodas bus atsektas ir koreguotas programišių naudai. Apsaugos, esančios programiniame kode efektyvios yra tik tuomet, kai įsilaužimo kaina yra didesnė, nei pačios programinės įrangos, kadangi kiekviena apsauga yra tik trukdis programišiui pasiekti jį dominantį turinį. Efektyvi apsaugos priemonė gali paskatinti įsigyti legalią programinę įrangą.

Programinio kodo apsaugojimas yra sudėtingas uždavinys, tam reikia išanalizuoti daug įvairiausių metodų, algoritmų, reikia patikrinti jų patikimumą, juos papildyti ar pakoreguoti ir pritaikyti savo programiniame kode. Daug metodų yra tiesiog nedokumentuojami dėl jų patikimumo sumažėjimo, žinodamas kaip yra saugojama programinė įranga, programišius galėtų susitelkti tik į tą vietą.

1.1. Programinio kodo maskavimas

Dažniausiai naudojamas ir vienas iš efektyviausių būdų kaip apsaugoti programinį kodą yra pačio programinio kodo skaitymo apsunkinimas. Kodo maskavimas yra būdas kuomet programiniame kode naudojami metodai, kintamieji, klasės yra pervardinami ir sumaišomi pagal tam tikrą struktūrą ir užšifruojami, tačiau pats vykdymo procesas nekinta.

Maskavimas apsunkina įprastą programos skaitymą, ir negali būti atkurtas į pirminį kodą. Toks būdas naudojamas apsisaugoti kodą nuo įprasto atkūrimo, programišiui tokį kodą sunku suprasti, o norint perprasti ir išsiaiškinti naudojamus algoritmus ir saugos mechanizmus, tai gali pareikalauti daug žmogiškojo ištekliaus, pinigų ir kompiuterio energijos. Pažvelgus iš priešingos pusės, programišius gali užmaskuoti virusinio tipo programinę įrangą, tam jog būtų sunku aptikti joje naudojamus kenkėjiškus metodus. Naudojant maskavimą neišvengiamai atsiranda neigiamos pasekmės, tai, kad kodas tampa didesnis ir lėtesnis, todėl programinės įrangos kūrėjas turėtų nuspręsti ar jam verta taip elgtis, ir naudoti kodo maskavimą, ir būtent kokį kodo maskavimo įrankį naudoti. Maskavimo programinės įrangos kūrėjai giriasi jog jų algoritmų niekas nežino ir todėl saugumo lygis yra aukščiausias. Anot Microsoft (Poulsen, 2015), uždaras kodas yra daug saugesnis, nei atviras kodas, nes lengvai prieinamas kodas leidžia ieškoti pažeidžiamumų. Tačiau tiesa yra ta, jog programišiui nėra neįmanoma rasti programinio kodo pažeidžiamumų paminėtose atvejuose, tai yra tik laiko klausimas (Christian Collberg J. N., 2009). Taigi tam jog suprasti, kaip viskas vyksta, žemiau pateikiamas pavyzdinis programinio kodo variantas (1.1. Pav.).

```
class Program
{
    static void Main(string[] args)
    {
        int a = 0;
        int b = 1;
        int c = 2;
        while (c * b != a)
        {
            Console.WriteLine("c * b = " + c * b);
            a++;
        }
    }
}
```

1.1. Pav. Originalus kodas

Kodas yra paprastas, jis atvaizduoja du kartus skaičių „2“. Šis kodas toliau yra užmaskuojamas pasinaudojant „CryptoObfuscator“ programiniu įrankiu, kuris yra lengvai valdomas, dažnai naudojamas dėl prieinamos kainos tiek įmonėms, tiek individualiems programuotojams (Software, 5 Reasons You Should Buy Crypto Obfuscator Now , 2015). Akivaizdu jog kodas tapo sunkiau nuskaitomas (1.2. Pav.) ir pateiktame maskuoto kodo pavyzdyje yra tik pagrindinė kodo dalis, kadangi buvo sukurtos dar atskiros keturios klasės. Originalus kodas sudaro viso labo tik 14 eilučių, tačiau panaudojus maskavimo įrankį, eilučių kiekis išaugo iki 547, nors dalis kodo yra ir tikrinama programos licencija, nes buvo naudojama bandomoji maskavimo įrankio versija, tačiau tai naudoja papildomus kompiuterio resursus. Taigi galime įsivaizduoti, kaip pasikeis programinis kodas, jei jame bus tūkstančiai ar šimtai tūkstančių kodo eilučių, ir kiek, tai užtruks laiko programišiui, kol supras jo veikimą. Toks maskavimo būdas nėra labai priimtinas variantas, kadangi kriptografiniai metodai visados slepia ilgą raktą, kurį galima atrasti.

```
internal class c87e3835053b5666f4391dc88e2bd248c
{
    private static void c05573d4f8f908e5726664c40aeb02009(string[] c7ec9289e51bb692feab38fd5b44675f2)
    {
        c9b6d9fdd343a534bb5d719b63232a825.c73c8d8978dfccf696f95d15cec110924();
        int num = 0;
        int num2 = 1;
        int num3 = 2;
        while (num3 * num2 != num)
        {
            Console.WriteLine("c * b = " + num3 * num2);
            num++;
        }
        while (true)
        {
            switch (7)
            {
            case 0:
                continue;
            }
            break;
        }
        if (!true)
        {
            RuntimeMethodHandle arg_4B_0 =
                methodof(c87e3835053b5666f4391dc88e2bd248c.c05573d4f8f908e5726664c40aeb02009(string[])).MethodHandle;
        }
    }
}
```

1.2. Pav. Užmaskuotas kodas

1.1.1. Apgrąžos inžinerija

Apgrąžos inžinerijos pagalba mes galime atkurti veiksmų sekas, metodus, kurie naudojami programinėje įrangoje, tai įprasto lengvai skaitomo ar užmaskuoto kodo pavertimas į mašininį kodą, kuris sudarytas iš dvejetainio kodo(0 ir 1). Jie yra siunčiami į loginį[1] procesorių, ir yra atliekami tam tikri aprašyti veiksmai. Tokiu būdu galima išstudijuoti kaip programa veikia, kokia jos greitaveika ir kokių klaidų joje yra, kurių negalima pastebėti pirminiame kode. Dažnai tam, kad būtų lengviau skaityti dvejetainį kodą, yra naudojami šešioliktainio kodo įrankiai. Jų pagalba galima nustatyti tam tikras programos vietas, ir matyti kaip jos veikia. Programišiai tam, kad įsilaužtų ar kitaip pakenktų programinei įrangai naudoja „disassembler“ - įrankį, skirta skaityti dvejetainį kodą ir pateikti vykdomąjį kodą teksto forma - asemblerio kalbą (Rouse, Reverse Engineering Definition, 2015) (1.3. Pav.).

```
            drow["1"] = 10;
000001eb  mov         rax,12283060h
000001f5  mov         rax,qword ptr [rax]
000001f8  mov         qword ptr [rsp+000000B8h],rax
00000200  mov         rax,qword ptr [rsp+30h]
00000205  mov         qword ptr [rsp+000000C0h],rax
0000020d  mov         dword ptr [rsp+38h],0Ah
00000215  mov         rcx,7FFBEC32F060h
0000021f  lea         rdx,[rsp+38h]
00000224  call        000000005F8C4C60
00000229  mov         r8,rax
0000022c  mov         rax,qword ptr [rsp+000000C0h]
00000234  cmp         byte ptr [rax],0
00000237  mov         rcx,qword ptr [rsp+000000C0h]
0000023f  mov         rdx,qword ptr [rsp+000000B8h]
00000247  call        000000005B03D360
0000024c  nop
```

1.3. Pav. Asemblerio kalba vykdomas priskyrimas eilutei

Programuotojai gali tyrinėti virusinio tipo programinę įrangą, matyti kaip ji dirba iš vidaus ir kokiais pažeidžiamumais ji naudojasi, tokiu būdu galima apsaugoti naujai kuriamus produktus. Yra keturi scenarijai, kurie susiję su programinės įrangos saugumu panaudojant apgrąžos inžinerija (Eilam, 2005):

- Virusų aptikimas ir jų šalinimas.
- Metodų nuo apgrąžos inžinerijos testavimas.
- Kriptografinių metodų testavimas nuo pažeidžiamumų.
- Dvejetainio kodo saugos tikrinimas.

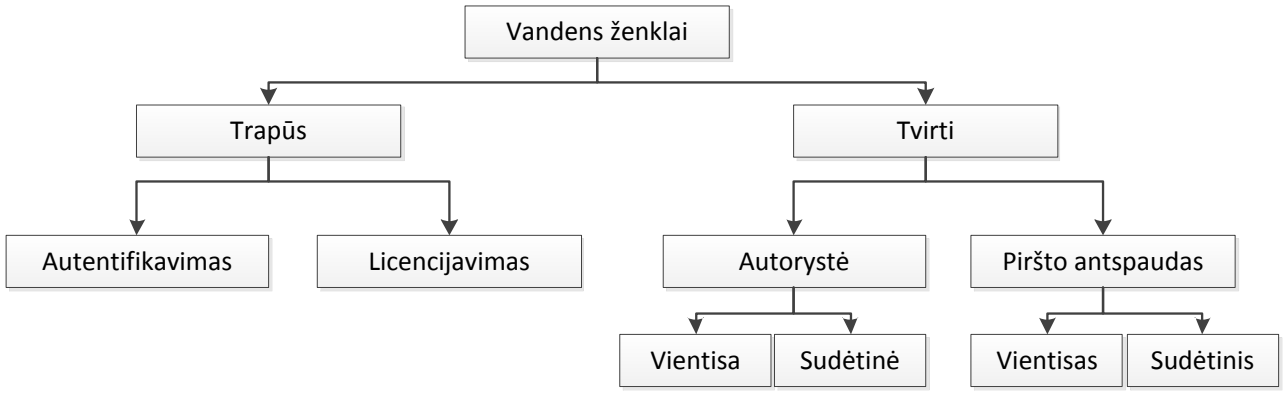
Daugelis programinės įrangos kūrimo įrankių sugeba mašininį kodą perversti į asemblerį, kadangi egzistuoja ryšys vienas su vienu vykdyti mašininio ir asemblerio kodų instrukcijas (Irvine, 2006). Taigi kitas žingsnis yra paversti asemblerio programinį kodą į aukštesnės programavimo kalbos kodą, tokiu būdu kodas taps lengviau skaitomas, tačiau jis nebus toks koks buvo pradžioje, nes aukšto lygio programinis kodas virsdamas į mašininį praranda didžiąją dalį savo informacijos. Todėl tokiu būdu mėginti atsekti koks kodas buvo rašytas programuotojo yra labai sudėtingas uždavinys. Teoriškai yra įmanoma sukurti gerą dekompiletorių, nors nemažai ekspertų ir nesutinka su tuo todėl, kad dalis informacijos negrįžtamai yra prarandama kompiliavimo metu (Eilam, 2005). Yra vienas atviro kodo

[1] Tai gija atliekanti tam tikras instrukcijas, loginis procesorius yra fizinio procesoriaus dalis

projektas, kuris tikima vieną dieną sugebės atkurti mašininį kodą į pirminį, visiškai tokį koks ir reikalingas, pavadinimu „Boomerang“ (Boomerang decompiler, 2015).

1.1.2. Vandens ženklai

Programinis vandens ženklų naudojimas yra dar vienas būdas, apsaugoti savo programinį kodą nuo vagystės, tai nėra priemonė, kuri sutrukdys programišiui pasisavinti kodą, tai priemonė, kuri gali priversti jį susilaikyti nuo tokio pasikėsinimo. Vandens ženklinimas programinėje įrangoje, tai unikalaus identifikatoriaus įdėjimas į kodo dalį, kuris yra paslepiamas ir po tam tikro laiko jį galima išgauti, tai tampa įrodymu į programinės įrangos ar jo dalies nuosavybę (Myles, 2004). Priklausomai nuo to kaip bus saugoma programinė įranga, taip turėtų būti naudojami ir vandens ženklai. Juos galima naudoti dvejopai. Pirmiausia atsparūs transformacijoms, kas reiškia, kad naudojant kodo maskavimo įrankius, ši vieta turi būti nepakitusi, tam, kad programinės įrangos kūrėjai galėtų aptikti ir įrodyti nuosavybę į kodą. Antra, atvirkščiai, turi būti neatsparūs, kas reikštų jog bet koks programinės įrangos pakitimas gali išjungti vandens ženklus, tai ypač naudinga jei programinė įranga naudoja licencijavimą (Jasvir Nagra, 2001). Žemiau pateikiamas vandens ženklų taksonomijos medis (1.4. Pav.).



1.4. Pav. Vandens ženklų taksonomijos medis (Jasvir Nagra, 2001)

Vandens ženklo patikimumas priklauso nuo to, kiek programišius užtruks laiko kol ras ir pašalins vandens ženklą, nesugadinus originalaus pirminio kodo, todėl reikėtų pasirūpinti, jog vandens ženklas būtų gerai paslėptas. Priešingai nei programišiui, programinės įrangos kūrėjui, toks vandens ženklas turėtų būti lengvai aptinkamas, tačiau tai padaryti yra pakankamai sudėtinga, todėl yra naudojami žymekliai, kurie parodo, kur yra naudojamas vandens ženklas, tačiau tokiu būdu programišius žymeklį taip pat gali rasti, o su juo ir vandens ženklą (Christian Collberg G. M., 2003).

Vandens ženklų technikos gali būti tiek statinio kodo, tiek dinaminio. Statinio kodo vandens ženklai gali būti naudojami pačiame vykdomajame faile, ar duomenų failuose, paveikslėliuose, teksto failuose ir panašiai. Tokiu būdu naudojami vandens ženklai yra neatsparūs kodo maskavimui, kaip ir kiti statiniai duomenys, o ,tai gali išsikreipti juos, kas reiškia jog vandens ženklai tampa neveikiantys (Christian Collberg C. T., 1999). Dinaminio kodo vandens ženklai yra naudojami programos paleidimo būsenoje, ir rečiau – programos kode. Duomenų struktūrose naudojami dinaminiai vandens ženklai gali būti taip pat neatsparūs kodo maskavimui, dėl įvairiausių maskavimo algoritmų, kurie gali negrįžtamai sugadinti vandens ženklus. Žemiau pateikiamas statinio ir dinaminio vandens ženklo pavyzdys (1.5. Pav.).

```
static void Main(string[] args)
{
    int c = 1; //
    int a = 2; //Paslėptas vandens ženklas
    int b = 3; //
    Console.WriteLine("Copyright by Donatas"); //Matomas vandens ženklas
    Console.WriteLine(a+b+c);
}
// Statinis vandens ženklas
```

```
static void Main(string[] args)
{
    int a = 1;
    int b = 2;
    int c = 3;
    int d = b * a * c * 100;
    Console.WriteLine(a+b+c);
}
// Dinaminis vandens ženklas
```

1.5. Pav. Statinis ir dinaminis vandens ženklas

Iš pateikto pavyzdžio matome jog statinis vandens ženklas gali būti matomas, panaudojus unikalų tekstą, tačiau jis nėra atsparus kodo maišymui po kurio tas tekstas gali pavirsti atsitiktinų simbolių kratiniu, ir jei nėra galimybės atkurti pirminio kodo, toks vandens ženklas yra visiškai sugadinamas, tačiau tai praverčia jei programinis kodas yra šifruojamas tam tikru algoritmu, ir vėliau yra galimybė atsišifruoti. Paslėpto statinio vandens ženklo pavyzdys yra ne iš eilės sudėlioti kintamieji, kurių eiliškumą žino tik programinės įrangos kūrėjas. Šitas variantas taip pat gali būti neatsparus kodo maskavimo įrankiams, kadangi tiek funkcijų, tiek kintamųjų vardai yra pakeičiami numatytais pavadinimais. Dinaminio vandens ženklo variantas, yra naujas kintamasis, kuris apskaičiuojamas pagal tam tikrą eilę kintamųjų ir panaudojus papildomą nekintamą dydį, ir jei kodo maskavimo įrankis pakeis kintamųjų vardus, tačiau eilė ir papildomas kintamasis liks tie patys.

1.2. Programinės įrangos licencijavimas

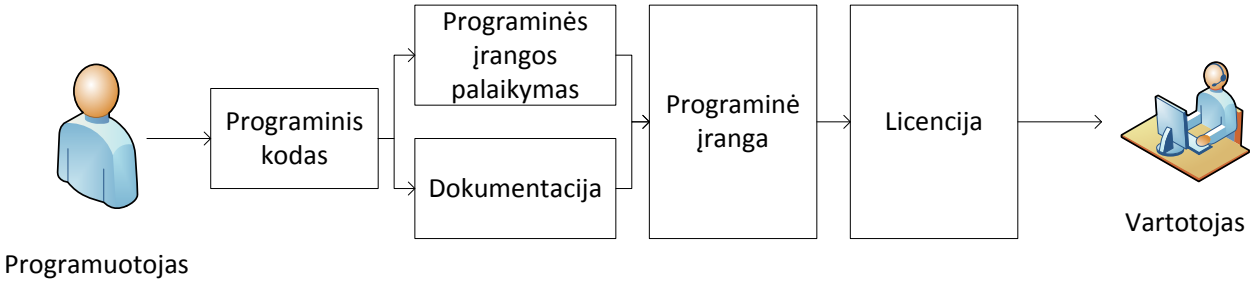
Programos licencija yra vartotojo teisė į programinę įrangą, pagal autoriaus pateiktą aprašymą. Licencijos gali būti uždaros, kuriose programinės įrangos kodas nėra viešai prieinamas ir atviro kodo, kai programinis kodas yra prieinamas viešai, ir gali būti modifikuotas kiekvieno asmens. Tokios licencijos toliau skirstomos į bandomąją, mokamą, nemokamą ir piratinę.

Licencijos aprašymas dažniausiai yra pateikiamas su programos dokumentacija, ir vartotojas prieš pradėdamas naudotis programine įranga, turi sutikti su licencijoje aprašomais punktais. Vienas iš punktų gali būti, tai jog programinės įrangos kūrėjai neatsako už nelegalios programinės įrangos veikimą. Šis punktas programinės įrangos autoriui suteikia teisę keisti programos veikimą, jei joje yra aptinkamas nelegalus vartotojas. Aptikimui gali būti naudojamos įvairios priemonės, viena iš jų – licencijos failo patalpinimas programos kode arba atskirame faile. Neretai dabar sutinkama programos licencija būna patalpinta serveryje, ten gali būti laikomas vartotojo raktas, prisijungimas su slaptažodžiu ir panašiai. Viso to priežastis, dažnai beatsinaujinanti programinė įranga, kuriai reikalinga interneto prieiga. Serveryje laikomu kodo fragmentu pridedame papildomą apsaugą, programišius negali pasiekti jame esančio kodo, į jį neįsilaužus, tačiau gali tikrinti įeinančius ir išeinančius rezultatus. Licencija naudojanti raktinius kodus yra paremta generavimo algoritmu, ir suradus, ar iššifravus tą algoritmą, galima sukurti tokių kodų generavimo įrankį, kurį paviešinus, jis pridarytų labai daug žalos programinės įrangos autoriui. Todėl reikėtų

labiau pasirūpinti programinės įrangos apsauga ir tik fiktyviai apsaugai naudoti metodus, kurie plačiai naudojami.
Programinės įrangos licencijos tipai skirtos vienam vartotojui (Services, 2015):

- Pilnoji licencija – neribotas programos naudojimas, su garantiniu palaikymu.
- Abonementinė licencija – neribotas programos naudojimas ribotam laikui.
- Laisvai platinama – nemokama ir neribota programinė įranga, gali būti su išimtimis.
- Bandomoji licencija – tai ribotą laiką ar ribotą funkcionalumą leidžianti licencija.

Sukurti teisingą licencinį produktą yra sudėtingas uždavinys. Programinės įrangos kūrėjas turi netik išmanyti programavimo technikos, tačiau ir tam tikrų verslo procesų, tokių kaip marketingas, pardavimai, bendravimas su klientu ir panašiai (Halina Kaminski, 2005). Visi šie žingsniai yra susiję. Žemiau pateikiamas procesas kaip programinė įranga pasiekia vartotoją (1.6. Pav.). Kadangi programinė įranga dažniausiai būna sukurta pilnos struktūros, todėl tik licencija apsprendžia, kokio funkcionalumo vartotojas ja naudosis.



1.6. Pav. Procesas tarp programuotojo ir vartotojo

Licencijos failas yra vienas iš būdų, apsaugoti programinę įranga, kuri skirta naudotis be interneto ryšio, tačiau jis taptų reikalingas licencijai atnaujinti. Vartotojas pasirašydamas ar sutikdamas su licencijos nuostatomis, neretai turi užregistruoti savo duomenis, tokius kaip vartotojo vardas, adresas, kontaktinė informacija. Gaudamas programinę įrangą vartotojas su ja gauna ir lokalų licencijos failą. Faile gali būti saugoma registracinė informacija, data kada prasidėjo ir kada turi baigtis licencijos galiojimas, jei tai bandomoji ar abonementinė licencija, ir kita nuo programuotojo sprendimo priklausanti informacija. Tokie failai būna užšifruojami pagal tam tikrą šifravimo metodiką, kad paprastas vartotojas negalėtų lengvai keisti licencijos faile esančių duomenų. Šifravimo raktas turėtų būti pasiskirstęs kažkur programiniame kode, ir sudarytas iš kelių dalių, kur panaudojus algoritmą jis būtų gautas (Mohab U. AbdelHameed, 2009). Nesvarbu iš kiek dalių bus sudarytas raktas, jis neturėtų būti pažeidžiamas kodo maskavimo, jei tokie įrankiai bus naudojami.

1.2.1. Šifravimo metodai

Norint apsaugoti svarbų turinį, reikia jį užšifruoti, tačiau to nepakanka, šifravimas turi būti efektyvus, kad bet kuris panorėjęs asmuo nesugebėtų iššifruoti svarbių duomenų. Yra trys pagrindiniai šifravimo tipai (McDonald, 2010):

- Simetrinis šifravimas.
- Asimetrinis šifravimas.

Simetrinis šifravimas naudoja vieną slaptą raktą, kuriuo yra užšifruojamas tekstas ir tuo pačiu raktu yra atšifruojamas (1.7. Pav.)

1.7. Pav. Simetrinis šifravimas

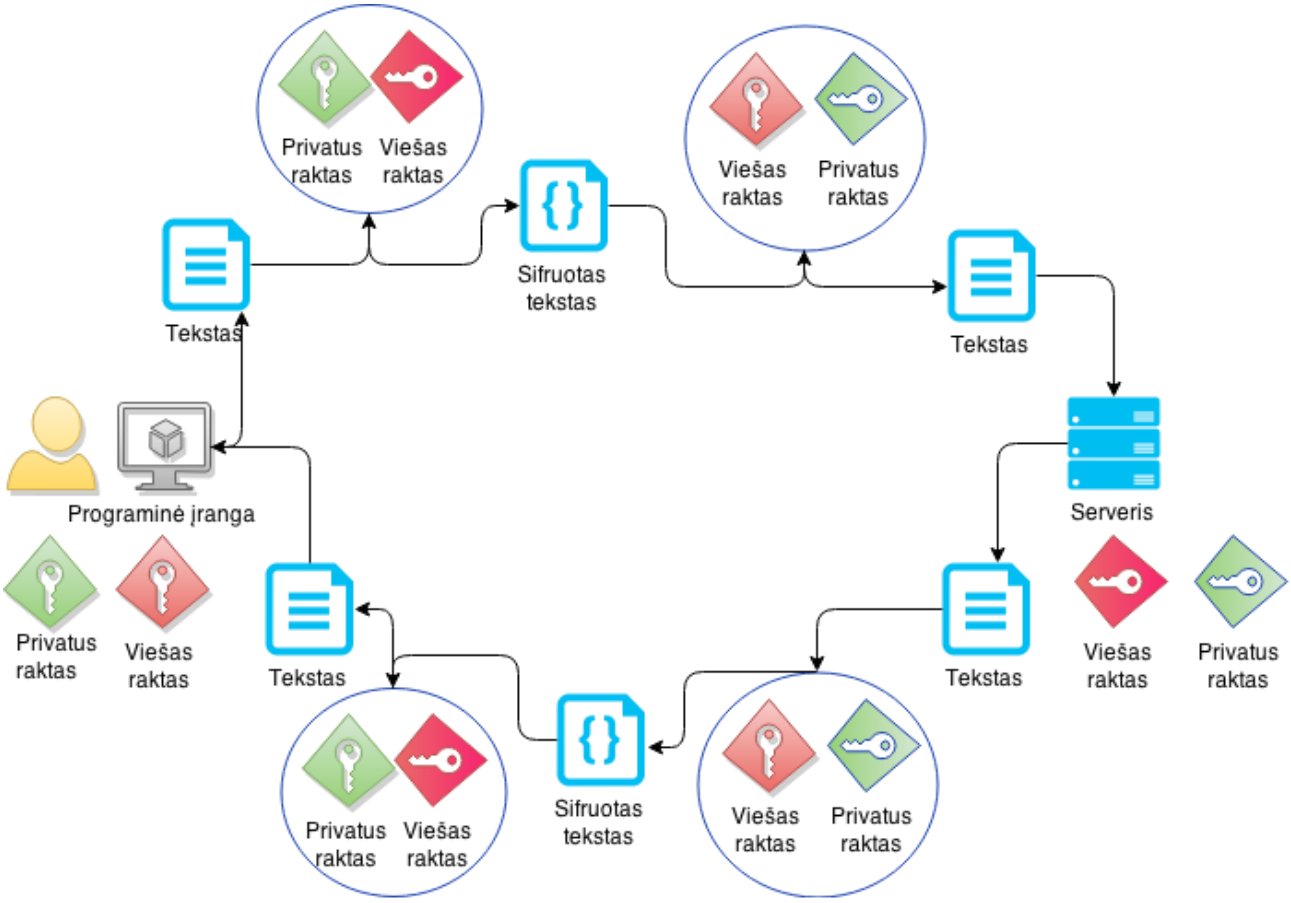
Tai paprastas būdas užšifruoti tekstą, panaudojant tą patį slaptą raktą užkuoduoti ir programiniame kode jį paslėpti. Populiariausiai simetrinio šifravimo algoritmai, kurie buvo sukurti praeito amžiaus pabaigoje (maldevel, 2015) (1.1. Lentelė).

2.1. Lentelė Populiariausiai simetrinio šifravimo algoritmai

Algoritmas	Paskelbta	Rakto dydis bitais	Šifravimo tipas	Bloko/srauto dydis bitais	Pakartojimai
AES/Rijindael	1998	128;192;256	Blokinis	128	10;12;14
Blowfish	1993	1-448	Blokinis	64	16
CAST5	1996	40-128	Blokinis	64	12;16
IDEA	-	128	Blokinis	64	8,5
RC4	1994	40-2048	Srautinis	2064	256
Serpent	1998	128;192;256	Blokinis	128	32
3DES	1998	56;112;168	Blokinis	64	48
Twofish	1998	128;192;256	Blokinis	128	16

Tačiau šiuo metu labiausiai žinomas AES/Rijindael yra vienas iš mažiausiai pažeidžiamų algoritmų, yra greitas ir saugus (Mansoor Ebrahim, 2013).

Asimetrinis šifravimas dar kitaip vadinamas viešo rakto šifravimas (Public-key cryptography, 2015) dažniausiai naudojamas keičiantis informacija, tai gali būti tekstinė žinutė, ar internetinis puslapis. Asimetrinis šifravimas vyksta naudojant raktų poras, tai reiškia jog sąryšį tarp dviejų programų sudaro išvis 4 raktai. Kiekviena iš pusių turi po du raktus, viešąjį ir privatų raktą. Žemiau pateikiama kaip vyksta asimetrinis šifravimas (1.8. Pav.)



1.8. Pav. Asimetrinis šifravimas

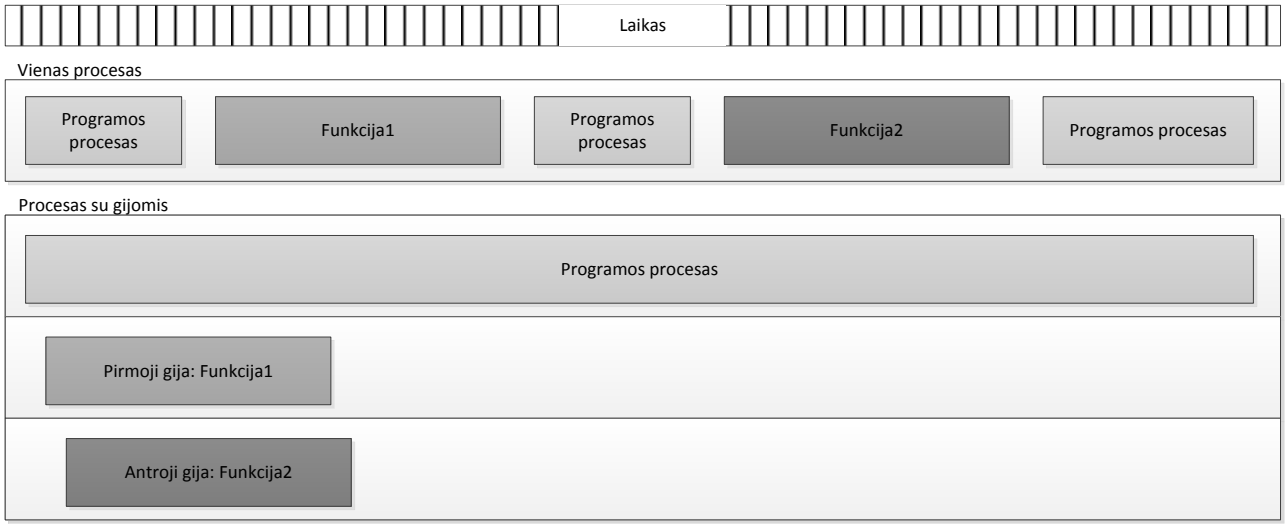
Asimetrinis šifravimas vyksta naudojant privatų ir viešą raktą, iš kurių susidaro slaptas raktas, panašiai kaip simetriniame šifravime, tik skirtumas toks, jog užšifruojama su viena pora raktų, o atšifruojama jau su kita. Parodytame paveiksle (1.8. Pav.) vyksta sąryšis tarp serverio ir vartotojo. Vartotojas naudojasi tam tikra programine įranga, kurioje yra serverio viešas raktas, jis pasiunčia tam tikrą užklausa ir ta užklausa yra užšifruojama vartotojo privačiu raktu, ir serverio viešuoju raktu, o serveryje jau atšifruojama pasinaudojant vartotojo viešuoju, ir serverio privačiuoju raktais.
Dažniausiai sutinkami asimetrinio šifro algoritmai (Decryption, 2015):

- RSA
- Diffie-Hellman
- DSA
- ElGamal

Vienas iš labiausiai naudojamų šiuo metu yra RSA asimetrinis šifravimas, kuris gali būti naudojamas ir kaip šifravimo algoritmas, ir kaip skaitmeninis parašas (Decryption, 2015). Naudojamas rakto dydis yra nuo 1024 iki 4096 bitų.

1.3. Gijos

Gija, tai nepriklausomas nuo programos srautas, kuris gali vykti atskirai (Yadav, 2015). Gijų gali būti labai daug, tai vadinama daugiagijiškumu. Programos metu yra paleidžiamas procesas, kuris vykdo programą, tam yra skiriama dideli kiekiai resursų, o gija tai tas pats procesas, tačiau nereikalaujantis daug resursų, kad atlikti jai paskirta užduotį. Tai gali pasitarnauti jei programos vykdymo metu vyksta tam tikri duomenų registravimai ar tikrinimai sistemoje, nes vartotojas gali naudotis programa, o tuo pat momentu vyksta kiti procesai, kurie nėra pastebimi. Vieno proceso ir proceso su gijomis palyginimas pateiktas žemiau (1.9. Pav.).



1.**9. Pav. Vieno proceso ir proceso su gijomis palyginimas**

Tam, kad būtų galima panaudoti daugiagijiškumą, reikalinga šią galimybę palaikanti operacinė sistema. Procesoriaus laikas yra skirstomas tarp procesų ir gijų, nuo užduoties prioritetų priklauso kiek užduotis bus vykdoma, ir pereinama prie kitos užduoties.

Naudojant gijas mes programinės įrangos apsaugoti negalime, tačiau gijas galima panaudoti kreipiantis į failus, kurių pavadinimai yra tokie pat, bet skiriasi jų saugos kodas. Tokia galimybė atsiranda, jei programinė įranga bus naudojama Microsoft „Windows“ operacinėse sistemose, kurių failų sistema yra NTFS (Microsoft, 2015). Tokiu būdu naudojant identiškus duomenis įrašymui, tikėtina jog programišius nepastebės paslėpto failo.

1.4. Programiniai įrankiai

Toliau pateikiami statistiniai duomenys apie operacines sistemas ir programavimo kalbas. Trumpai apžvelgiami kodo maskavimo įrankiai.

1.4.1. Operacinė sistema

Trumpas operacinių sistemų palyginimas skirtas apžvelgti kaip jos yra plačiai naudojamos ir kiek pažeidžiamumų yra fiksuojama. Operacinių sistemų pažeidžiamumas gali įtakoti ir kuriamos programos pažeidžiamumą, dėl galimybės prieiti prie tam tikrų atminties blokų, virtualizuoti tam tikrus procesus, kurie išderintų programą ir panašiai. Pagal statistinius duomenis (Vaughan-Nichols, 2015), 2015 metais populiariausia operacinė sistema yra Windows, kuri užima daugiau kaip 55% ir (Khandelwal, 2015) 2014 metų duomenimis yra mažiausiai pažeidžiama operacinė sistema (1.3. Lentelė).

1.4. Lentelė Personalinių kompiuterių operacinių sistemų palyginimas

Operacinė sistema / Kriterijai	Rinkos dalis (2015m. vasario m.)	Pažeidžiamumų kiekis (2014m.)
Mac OS X	17.21%	147
Linux kernel	22.56%	119
Windows	55.51%	30-38[2]

1.4.2. Programavimo kalba

Remiantis statistiniais duomenimis (BV, 2015) pačios populiariausios paskutinių 5 metų programavimo kalbos yra:

- Java
- C
- C++
- Objective-C
- C#

Kaip augo programavimo kalbų populiarumas paskutiniais metais palyginimas pateikiamas žemiau (1.5. Lentelė).

1.6. Lentelė Labiausiai naudojamos programavimo kalbos

	2011-01	2012-01	2013-01	2014-01	2015-01	Pokytis
Java	17,77	17,47	17,41	16,51	15,52	-12,67
C	15,82	16,97	17,85	17,94	16,70	5,56
C++	8,78	8,05	9,14	7,56	6,70	-23,70
Objective-C	3,01	6,91	10,28	11,09	6,95	230,89
C#	6,22	8,76	6,19	5,85	5,04	-18,98

1.4.3. Maskavimo įrankiai

Kodo maskavimo įrankių internete galima rasti tikrai daug, vieni jie platinami nemokamai, kiti mokamai, tačiau su galimybe išmėginti bandomąją versiją. Keleto tokių įrankių sąrašas (List of obfuscators for .NET, 2015):

- „Babel Obfuscator“
- „Crypto Obfuscator“
- „ConfuserEx“
- „Obfuscar“
- „Skater“

Kiekvienas iš jų bus trumpai detalizuotas pagal jų aprašymus, išbandyti naudojant paprastus algoritmus, tikrinama pasikeitusi programos greitaveika ir vienas iš jų pasirinktas tolimesniems etapams. Visi jie yra skirti .NET platformai.

„Babel Obfuscator“ yra mokamas įrankis, kuris palaiko skirtingomis kalbomis rašytą programinį kodą (pvz. C# ir C programavimo kalbos kartu). Pakeičia visų metodų, nustatymų, tipų ir

[2] Priklausomai nuo Windows versijos

klasių pavadinimus į unikodo simbolius, kurie yra neatvaizduojami. Metodai yra pervardinami tais pačiais vardais, tačiau naudojant skirtingus parašus, ir tai turėtų programišiui apsunkinti suprasti kodą panaudojus apgrąžos inžinerija. Bandymas sugražinti maskuotą kodą į pirminį kodą parašyta aukšto lygio programavimo kalba yra neįmanomas. Nekintamos reikšmės yra užšifruojamos, taip sukuriant papildomą apsaugos lygį. Vienos standartinės licencijos kaina 115$, įmonei skirta neribota versija 1250$, kiekvienos licencijos atnaujinimas į nauja versija kainuoja atitinkamai 59$ ir 499$. Yra ir bandomoji versija, kuri užmaskuoja kodą veikiantį 14dienų.

"Crypto Obfuscator" labai panašių galimybių įrankis, kaip ir „Babel Obfuscator", tačiau papildomai turi galimybę į kodą įdėti vandens ženklus, ir gali paslėpti norimus metodus nuo aptikimo. Taip pat šis įrankis turi galimybę optimizuoti programos greitaveiką. Vienos standartinės licencijos kaina 149$, 8 žmonių įmonei skirta licencija 2399$, kiekvienos licencijos atnaujinimas į naują versiją kainuoja atitinkamai 59$ ir 959$. Yra ir bandomoji versija, kuri veikia 20dienų.

„ConfuserEx" atviro kodo įrankis, kuris palaiko daugumą komercinio tipo apsaugų, tokių kaip metodo kreipinių slėpimas, jų šifravimas, apsaugos nuo programinio kodo derinimo, nekintamų reikšmių šifravimas. Šis įrankis turi nemokamą licenciją ir jį galima tobulinti, taip kaip norima, todėl jokių garantijų dėl saugumo nelieka.

„Obfuscar" dar vienas atviro kodo įrankis, tačiau turintis tik pagrindines kodo maskavimo galimybes, tokias kaip klasių, metodų ir kintamųjų pavadinimų pakeitimus, kodo struktūros sumaišymą. Tai vienas iš paprasčiausių aptariamų kodo maskavimo įrankių. Licencija nemokama.

„Skater" įrankis, kuris turi standartiškai nekintančių reikšmių šifravimą, sugadina galimybę maskuotą kodą gražinti į pirminį kodą, metodų ir klasių pavadinimų pakeitimas, gali sujungti programos failą su atskirų bibliotekų failais į vieną. Anot kūrėjų, pasinaudojus šiuo įrankiu, programa draus apgrąžos inžineriją. Yra mokama ir nemokama programinio įrankio versijos. Vienos standartinės licencijos kaina 79,99$, įmonei skirta neribota versija 1709,99$.

Toliau buvo sukurta programa, kurioje panaudotas paprastas šifravimo ir atšifravimo algoritmas, kuris buvo vykdomas 10kartų, o viskas pakartojama 100kartų ir išvedamas vidurkis, kiek programa užtruko. Buvo patikrinta, kaip pasikeitė kodo apimtis ir kodo skaitomumas, panaudojus programinę įrangą skirtą gražinti maskuotą kodą į pirminį.

1.7. Lentelė Kodo maskavimo įrankių palyginimas

	Greitis, sekundėmis	Kodo apimtis, eilutėmis	Skaitomumas	Kaina, $
Originalus kodas	1,80654425	115	Visiškas	-
Babel Obfuscator	1,81037570	169	Galima suprasti veikimo principą	115-1250
Crypto Obfuscator	1,81109216	368	Sunku suprasti kaip viskas veikia	149-2399
ConfuserEx	1,82381262	682	Visiškai kitokios struktūros kodas, sunkiai suprantamas	0
Obfuscar	-	-	-	0
Skater	-	-	-	0-1709,99

Analizės metu du kodo maskavimo įrankiai negalėjo būti patikrinti dėl iškilusių problemų. Įrankis „Skater" nemokamos versijos įrankis galimai turėjo kenkėjiško kodo ir buvo aptiktas antivirusinės programos. Kitas įrankis „Obfuscar" nepasileido. Kodo maskavimo įrankių patikrinimui buvo panaudota:

- Virtuali mašina su Windows 8.1 64bit OS (AMD FX-8150;2GB RAM).
- ILSpy nemokamas atviro kodo įrankis maskuotą kodą gražinti į pirminį.

Visi analizuoti kodo maskavimo įrankiai buvo naudojami nemokamos ir bandomosios versijos. Dalį papildomo kodo eilučių po kodo maskavimo sudarė licencijos tikrinimas, tai galėjo įtakoti ir programos greičio rezultatus. Lėčiausiai testuojama programa veikė po nemokamo įrankio „ConfuserEx“, tačiau jame buvo aktyvuota daugiau apsaugos priemonių, pagal nutylėjimą.

1.5. Analizės išvados

Atlikus analizę buvo pastebėta, jog į programinės įrangos apsaugą yra daug finansuojama, tačiau, tai neduoda užtikrintos naudos, o esami metodai ne visados išgelbėja programinę įrangą nuo nelegalaus naudojimo. Labiausiai naudojami ir dokumentuojami apsaugos būdai yra kodo maskavimas ir vandens ženklai. Šioje vietoje nuolatos vyksta tobulinimai, ir naudojamos sudėtingos matematinės formulės, kad kuo sunkiau programišiui būtų pasinaudoti programine įranga nelegaliai.

Dalis programinės įrangos yra licencijuojamos pasitelkus interneto prieiga, ir viskas vyksta modeliu „Vartotojas – Serveris“, kur vartotojas yra priregistruojamas prie sistemos. Lokalios vietos programos naudoja licencijos failą, ar metodą, kuris būna paslėptas programinės įrangos kataloge.

Visi naudojami saugos metodai neapsieina be kriptografijos ir šifravimo metodų. Įrankių, kurie geba maskuoti kodą yra pakankamai daug, tačiau nevisi yra patikimi, ir didžioji dalis tokių įrankių yra mokami. Vienas iš aprašytų kodo maskavimo įrankių buvo palaikytas virusu, ir nebuvo galimybės saugiai jį išbandyti.

Apsaugos metodas bus kuriamas Microsoft „Windows“ aplinkoje, nes pagal statistika tai yra viena iš saugiausių operacinių sistemų. Kodo maskavimui bus naudojamas „ConfuserEx“ įrankis.

2. PROGRAMINIO KODO APSAUGOS PRITAIKYMAS

Norėdami apsaugoti programinę įrangą nuo nelegalaus naudojimo, privalome į ją įdiegti tam tikrus apsaugos mechanizmus, kurie kodą padarys sunkiau suprantamą, ir padės aptikti, kada programa yra naudojama nelegaliai. Toliau bus pateikiami būdai, kuriuos panaudojus, mes apsunkinsime programinio kodo skaitomumą ir galėsime aptikti, kuomet mūsų programa yra naudojama nelegaliai, tačiau tai nebus visapusiška apsauga, kuria galima pilnai pasitikėti. Metodas bus sudarytas iš dviejų lygių ir keturių pakopų, kurių žinoma gali būti ir daugiau (2.1. Pav.). Eilės tvarka surašyta pagal programišiaus susidūrimą su kiekviena iš pakopų. Atliekant metodo pritaikymą programinėje įrangoje, eilės tvarka tampa priešinga.

2.10. Pav. Apsaugos metodo schema

Pirmojoje pakopoje programišius susiduria su užmaskuotu kodu, kuris buvo maskuojamas pasitelkus kodo maskavimo įrankį, kadangi jie dauguma turi vandens ženklų panaudojimą kode, jų atskirai neišskiriam. Įveikus šią pakopą, antroje vietoje yra licencijos failas, ir jo tikrinimas. Po šių dviejų pakopų įveikimo, programinė įranga yra pilnai paleidžiama vartoti, tai reiškia jog pirmo lygio apsauga skirta fiktyviai suklaidinti programišiui. Antro lygio apsauga yra sudaryta iš paprastų metodų, kurie nėra sudaryti iš sudėtingų algoritmų todėl sekančių pakopų metodai turi būti paslėpti, tačiau nesudėtingi. Trečioje pakopoje paslepiame programinės įrangos elementą, kuris sugeneruojamas atsitiktinoje vietoje, ir užvedus ant jo pelės žymeklį, jis įjungia tikrinimą, kuris gali tikrinti tiek licencijos failą, tiek programinės įrangos modifikacijas. Vietoj pelės žymeklio tai gali būti ir kitas įvesties įrenginio panaudojimo patikrinimas. Ketvirtoje pakopoje panaudojant laiko intervalų skaičiavimą, ir gijas, patikriname ar licencijos failas užima pagal formulę numatytą dydį panaudojant bitų sumos skaičiavimą, taip pat gijų pagalba fiksuojame programos naudojimo laiką. Laiko intervalo fiksavimas gali būti paleidžiamas ir trečios pakopos metu, nes jei bus paleidžiamas pradžioje, jis bus lengvai aptinkamas.

Šio apsaugos metodo tikslas yra parodyti, jog naudojant paprastus metodus, tačiau juos sujungiant vienoje programoje ir panaudojant sudėtingas apsunkinimo priemones tokias kaip kodo maskavimas fiktyviai apsaugai, galime apsaugoti programinę įrangą nuo nelegalaus naudojimo, neskiriant tam didelių išlaidų.

2.1. Programinio kodo skaitymo apsunkinimas

Programinio kodo saugumas priklauso nuo to kaip sudėtingai yra užmaskuoti jame esančių metodų, klasių, funkcijų ir kintamųjų pavadinimai, ir kreipiniai. Maskavimas turi vykti pačiame paskutiniame programinės įrangos kūrimo procese, kai programinis kodas yra ištestuotas, ir paruoštas pateikti vartotojui. Atsižvelgiant į poreikius, reikia pasirinkti sau tinkamą maskavimo įrankį, geriausia, jog tai nebūtų bandomosios licencijos įrankis, nes pasibaigus bandomajam laikotarpiui, toks įrankis gali sugadinti maskuojamą kodą, ir jis bus neveikiantis, arba nuolatos reikalauti įsigyti pilną versiją.

2.2. Licencijos failo įterpimas

Nemažai programinės įrangos naudoja licencijavimą ir kartu su pateikta programa yra prisegamas licencijos failas. Priklausomai nuo licencijos, gali reikėti kas tam tikrą periodą atnaujinti failą.

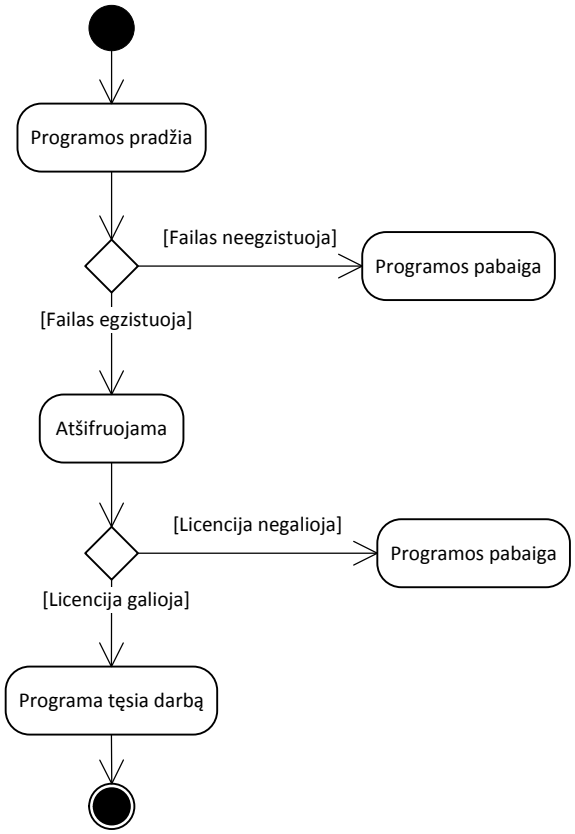
Licencijos failo įterpimas į programinę įrangą suteikia lankstumo tikrinant pačios programos nelegalų naudojimą, kadangi tai yra nedidelis failas, kuris gali laikyti svarbią informaciją ir būti naudingas tiek panaudojant fiktyvią apsaugą, tiek ir suveikus tikriems saugos mechanizmams. Licencijos failą turėtų sudaryti tam tikri vartotojo registracijos duomenys, licencijos numeris, data, galiojimo data jei naudojama ir panašūs duomenys, priklausomai nuo panaudojimo. Licencijos failas neturėtų būti lengvai nuskaitomas, ypač paprastam vartotojui, todėl tam reikia pakeisti kuriamo failo plėtinį, ir jo turinį užšifruoti. Visa tai skirta fiktyviam licencijos failo tikrinimui. Pagrindinį tikrinimą sudarys licencijos failo bitų sumos tikrinimas ir failo modifikavimo data. Tam jog nereikėtų kaskart atnaujinti programos, ir užtektų tik pakeisti licencijos failą, turime apsibrėžti tam tikrą sumą bitų, kurią naudosime licencijos faile. Apsibrėžus sumą, turime į failą pridėti nereikalingos informacijos tam, kad išlyginti bitų skaičių, nes, jei naudosime abonementinę, ar bandomosios versijos licencijas, faile kaskart atsiras datos ar kitokios informacijos pakitimai. Žemiau pateikiamas licencijos failo pavyzdinis variantas su originaliu ir šifruotu tekstu (2.2. Pav.).

```
000code = DO NOT EDIT THIS FILE OR THE LICENSE WILL NOT
WORK ANYMORE
CONGRATULATION YOU FOUND MY ENCRYPT KEY
User = Donatas Kvietkus (donatas.kvietkus@stud.ktu.lt)
expirydate= 2014-12-31
SN = 10023145678897
Windows = Windows 8.1
```

```
000code = DO NOT EDIT THIS FILE OR THE LICENSE WILL NOT
WORK ANYMORE
ffKJ2WQVqM/MToFThyH2CTqY/q92PAecgRJupwbJVspZZQ3bWk1zxG7
eiRhuOKU4KrAI21dy067bu7h3
c8uJJ4OxwaDKSETLd+fM20uKO5Y19rVgIT4P/E
+CekGfoiwFyM3quJgkg1SYBxEwnkxHTYNwcovqXTYz
KNC7RkA8UOjxY5O1KSdmFS91zUmCR5IWoHUfcegZSXO8j8XYC4tBYA=
=
User = Donatas Kvietkus (donatas.kvietkus@stud.ktu.lt)
yzTIjLAJojWwm1qxNlR8DHUmd0XvSuxCUbNKWdyLE8elZWUTrTnjqId
QDngthYF9dexEMgeQb+JQjqko
IG4Nyx03pVADozxtDIWA4HH5p0tMIL9mY83JrVo6SZYu2VIBKs9L12g
ytVOOgg01T7Q5kQ==
Windows = Windows 8.1
```

112. Pav. Licencijos failo pavyzdys

Licencijos failo skaitymo procesas vykdomas pradžioje - programos paleidimo metu, tuomet vyksta fiktyvus licencijos tikrinimas (2.3. Pav.). Failas yra nuskaitomas, atšifruojamas pagal užšifruotą raktą, ir patikrinama licencijos galiojimo data. Jei data yra mažesnė nei esanti dabartinė kompiuterio data, programa įspėja kad licencija yra pasibaigusi, ir išsijungia. Programa išsijungia taip pat jei nepavyksta rasti licencijos failo.



12 Pav. Licencijos tikrinimas

Praėjus tikrinimą, ir esant teigiamam rezultatui, programa yra paleidžiama pilnu funkcionalumu. Tikrinimas gali būti ir pakeistas programišiaus. Galimas scenarijus jog atradęs

tikrinimą programišius paduos jau teigiamus rezultatus, arba bandydamas pakenkti labiau, atsišifruos licencijos failą ir pakeis viduje esančius tikrinimo duomenis, tokius kaip licencijos galiojimo datą. Tokiu atveju programa pilnai pasileis ir veiks, tačiau programos veikimo metu, tam tikru momentu gali suveikti pagrindinis tikrinimo metodas, kuris kreipsis į atmintį ir tikrins licencijos failo turinio bitų skaičių, ir lygins su numatytu programos kūrimo metu. Taip pat galima tikrinti ir failo modifikacijas, kurios turėtų sutapti su failo sukūrimo diena, tačiau toks tikrinimas turėtų kreipinį į failą, kas gali būti lengvai pastebima. Tam jog nebūtų nereikalingų ir pastebimų kreipinių į failą, jį reikėtų užkrauti į atmintį. Antro lygio tikrinimas turi būti paslėptas, geriausiai tam tiktų susikurti atskirą biblioteką, kurioje vyksta įprasti procesai ar skaičiavimai, gali būti ir paleidžiama gija, kuri veikia tolygiai su programa, ir ją sunku pastebėti.

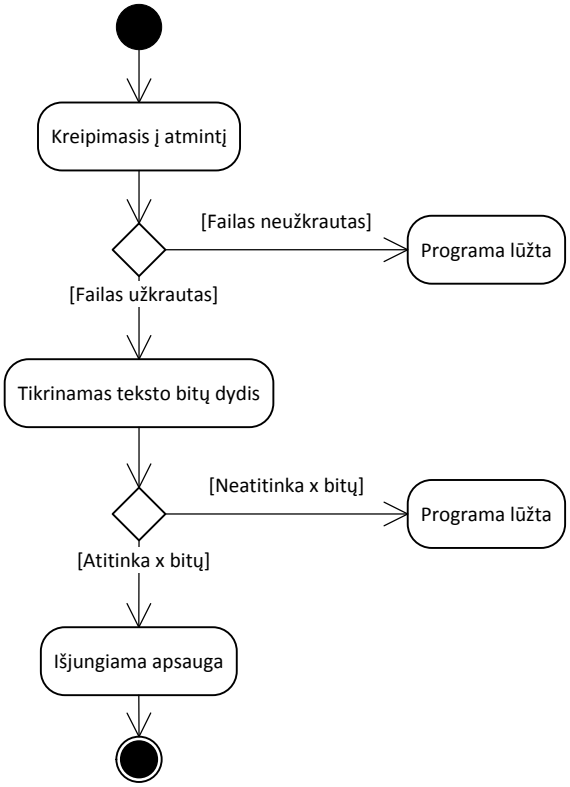
2.3. „Nematomo“ komponento panaudojimas

Didžioji dalis šiuo metu naudojamos programinės įrangos turi vartotojo aplinka, kuri turi grafinę sąsają. Taigi grafinę sąsają sudaro daugybė komponentų, tai mygtukai, tekstiniai laukai, kortelės, paveiksliukai, lentelės ir kt. Kiekvienas komponentas gali turėti tik jam skirtus įvykius. Esminis programišiaus suklaidinimo būdas yra paslėpti tikrinimo metodą būtent tarp komponento įvykių. Neretai programą sudaro daug įvairiausių bazinių klasių, kurios būna papildomos, ir tik tuomet atvaizduojamos vartotojui. Sukūrus vizualiai mažą komponentą bazinėje klasėje, ir paslėpus jį, galime būti tikri, jog tikrinimas vyks tik prie tam tikrų sąlygų, kas leistų suklaidinti programišių. Komponentas – gali būti nematomas mygtukas, kuris turėtų pelės žymeklio judėjimo fiksavimą, ir kai pelės žymeklis atsiduria ant komponento, programa vykdo tikrinimą, kreipiasi į atmintį ir tikriną licencijos failo turinį, o aptikusi jog programa vis tik yra naudojama nelegaliai pakeistų savo funkcionalumą, pavyzdžiui – programa pastrigtų. Tam, kad komponentas nebūtų taip lengvai susekamas, jis turėtų nuolatos būti generuojamas kitoje programinio lango pozicijoje, o įvykis paslėptas tarp daugybės nenaudingo kodo. Grafinė sąsaja su atsitiktinai generuojamu komponentu pateikta žemiau (2.4. Pav.)

13 Pav. Atsitiktinai generuojamas komponentas

Raudonai pažymėtas grafinis komponentas, mygtukas, sugeneruojamas atsitiktinai, jis yra nematomas, arba gali būti matomas, tačiau paslėptas už kitų komponentų. Jei mygtukas bus slepiamas už kitų komponentų, bus apribota mygtuko pozicija. Dar vienas būdas, mygtukas gali būti itin mažas arba susiliejąs su grafiniu langu.

Sukurtas ir paslėptas mygtukas turi aktyvuoti licencinio failo patikrinimą, kai tik ties juo atsiranda pelės žymeklis. Tikrinimas turi vykti jau į atmintį įkelto licencijos failo, tam kad nebūtų tiesioginių kreipinių, kuriuos lengva atsekti. Gali būti jog dalis kodo tapo modifikuota, ir programa jau nebetikrina licencijos failo, tuomet nebus ir atmintyje užkraunamų duomenų, tokiu atveju iškart gražinamas rezultatas jog programinė įranga yra modifikuota (2.5. Pav.). Kai gaunamas neigiamas rezultatas programa lūžta, tačiau galimi ir kiti scenarijai, tai priklauso ir nuo programinės įrangos kūrėjo. Vienas iš galimų scenarijų, tai programinio kodo funkcionalumo pakeitimas, toks kaip netikrų rezultatų atvaizdavimas, ar esamų rezultatų sugadinimas, tačiau prieš tai reikėtų įspėti vartotoją, jog programinės įrangos kūrėjai nėra atsakingi už nelegalaus naudojimo šalutinius padarinius.



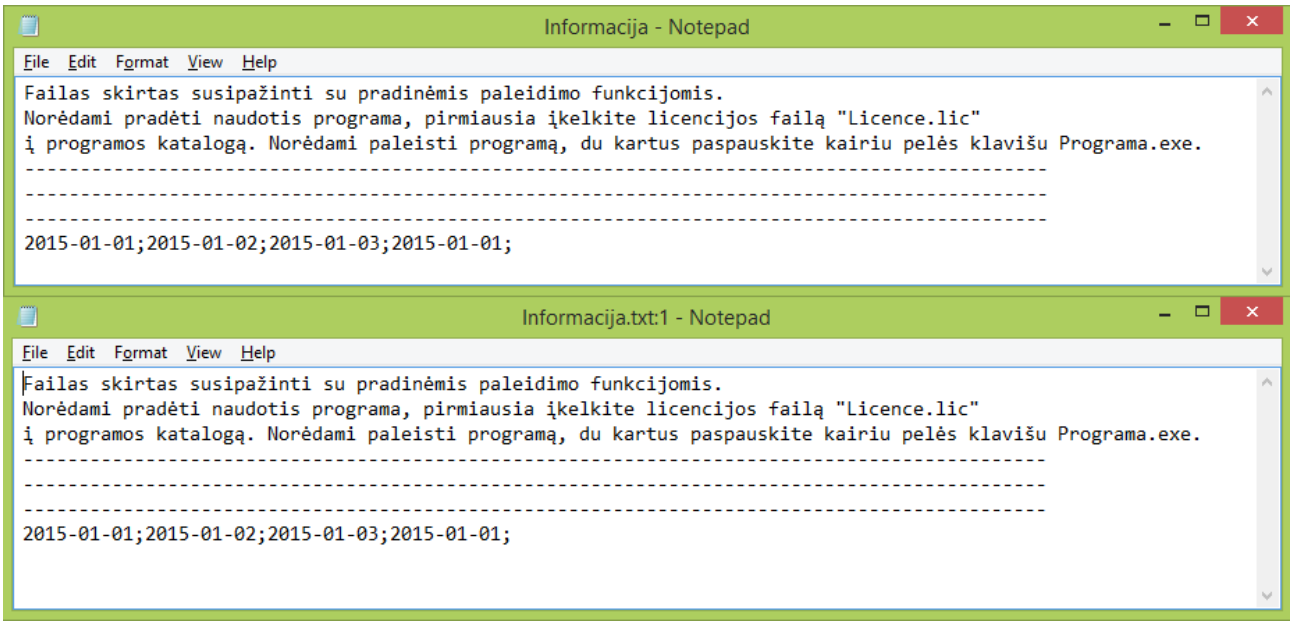
14 Pav. Paslėpto mygtuko tikrinimo metodas

Atlikus tikrinimo dalį reikėtų ją atjungti, dėl nereikalingo resursų švaistymo, žinoma toks sprendimas gali turėti ir blogąją pusę. Atjungus tikrinimą programa galima naudotis neribotą laiką, jei ji nebus paleista iš naujo. Paprasti vartotojai tikriausiai nuolatos išjungs programą, ar kompiuterį, tačiau programišius numatydamas jog egzistuoja toks apsaugos mechanizmas, gali programą įsirašyti į virtualiai sukurtą sistemą – virtualią mašiną. Tokiu būdu galima koreguoti sistemos laiką, ir niekados neišjungti programos, o koreguojant sistemos laiką, šis tikrinimas nebepadės, nes bandomoji licencija gali tęstis neribotą laiką, taigi atlikus tikrinimą, geriau jį atjungti. Norint programinę įrangą apsaugoti ir nuo laiko atsukinėjimo, tam bus panaudotas ketvirtos pakopos metodas – laiko žymė.

2.4. Laiko žymės tikrinimo modelis panaudojant gijas

Paskutinės pakopos saugos metodas skirtas fiksuoti laiką paslėptuose failuose, ar registruose. Jei naudojama failų sistema yra NTFS, kuri dažniausiai būna pagal nutylėjimą visuose Microsoft Windows operacinėse sistemose(nuo Windows NT), tuomet galime panaudoti duomenų paslėpimą, sukuriant paprastą tekstinį failą, kuris už savęs turės vieną ar daugiau tokio pat vardo failą tik su slaptu kodu. Tai nėra itin saugu, jei programišius žino kokie metodai yra naudojami, tačiau kaip

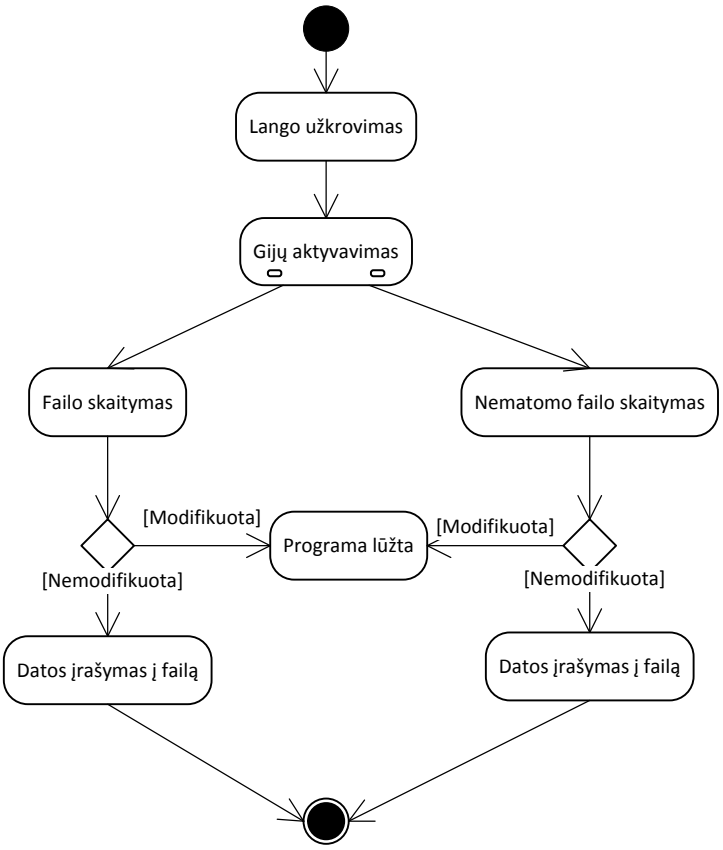
buvo minėta ankščiau, todėl ir nėra dokumentuojami individualūs saugos mechanizmai, kad nebūtų susikoncentruotą ties tikslia to metodo ar metodų vieta. Žemiau pateikiamas pavyzdys tekstinio failo ir už jo esančio paslėpto failo (2.6. Pav.)



15 Pav. Matomas ir nematomas failas

Svarbu jog abu failai turėtų kuo daugiau duomenų, kurie iš pirmo žvilgsnio būtų skirti paprastam vartotojui, kad nesukeltų įtarimo jog čia yra kažkoks tikrinimas, taip pat duomenys turėtų būti identiški, išskyrus fiksuojamus duomenis. Paskutinėje eilutėje žymima data, kada vartotojas prisijungė, iš pateikto pavyzdžio programa jau turėtų nustoti veikti, nes paskutinės dvi datos eina ne didėjimo tvarka, todėl programa tokį veiksmą turi interpretuoti jau kaip bandymą apgauti licenciją ir šiuo atveju vykdomas programos darbo nutraukimas, ar duomenų sugadinimas. Tokių nematomų failų gali būti ir daugiau, tačiau jei bus aptiktas pirmasis, tikėtina jog kiti bus rasti daug greičiau ir tai tampa neefektyvu.

Laiko fiksavimas turi būti paslėptas taip, kad jis įvyktų nebūtinai kiekvieną kartą kai programa yra paleidžiama, taip pat kaip mygtuko užvedimas su pelės žymekliu, dėl priežasties jog programišius tikrins visus procesus, kurie vyksta paleidimo metu, ir kas yra tikrinama iki programos pilno pasiruošimo darbui. Taigi laiko fiksavimas bus paslėptas viename iš programos langų, ir kai bus aktyvuojamas langas, bus paleidžiamos dvi gijos, kurios fiksuos prisijungimo informaciją į failą, taip pat tikrins jau esamus duomenis. Tam, kad gijos laisvai nebūtų aptiktos programoje, jas geriausia būtų paslėpti šifruotoje bibliotekoje, kuri būtų naudojama tam tikriems tos programos veiksmams, ar skaičiavimams atlikti. Vienas iš pavyzdžių būtų matematinių skaičiavimų biblioteka, kurioje atliekamos sudėtingos formulės ir tuo pačiu metu paleidžiamos gijos. Gijų veiklos diagrama (2.7. Pav.) vaizduoja kaip turi vykti tikrinimo ir datos fiksavimo metodai. Pavaizduotoje diagramoje naudojamos dvi gijos, tačiau esant papildomiems parametrams, tokiems kaip programinės įrangos laiko skaičiavimo fiksavimas, gali atsirasti ir daugiau gijų.

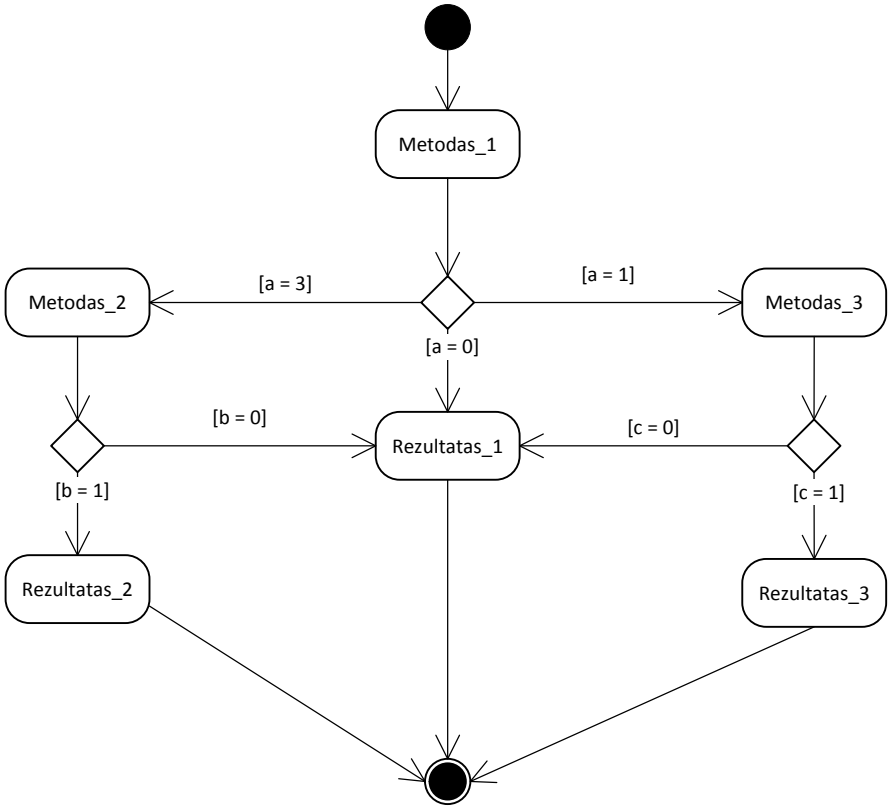


16 Pav. Gijų veiklos diagrama

Panaudojus datos ar laiko fiksavimą faile, mes galėsime nustatyti ar buvo mėgintą keisti kompiuterio datą, jei bus aptiktas matomas failas, ir jis koreguotas, tuo atveju bus panaudota nematomo failo informacija, ir programišius norėdamas atrasti, kur dar vyksta programos apsaugojimas, turės susidurti su nenaudingu kodu, kuris bus naudojamas visuose programos vietose, kad apsunkintų kelią iki esamų saugos metodų.

2.5. Nenaudingo kodo ir paspaudimo kombinacijų panaudojimas

Esant nedidelei programinio kodo apimčiai verta panaudoti papildomą kodą, kuris galėtų būti nenaudingas, arba naudingas iš dalies. Reikėtų naudoti kuo sunkiau skaitomus metodus, ir juos išmėtyti po visą programą, taip jog būtų sunkiau atsekti. Nenaudingas kodas programos veikimo greičiui neturėtų kenkti ir būti vykdomas tik tam tikrais atvejais, kuomet atsiranda numatyta veiksmų seka. Programišius sugaiš daug laiko, kol išsiaiškins visus metodus, nes tiek naudingo, tiek nenaudingo kodo metodai gali būti vykdomi tik įvykus tam tikrai veiksmų sekai, o tuo pačiu metu gali būti vykdomi kiti tikrinimo metodai panaudojant gijas. Žemiau pateikiama nenaudingo kodo veiksmų sekos diagrama (2.8. Pav.).



17 Pav. Nenaudingo kodo sekų diagrama

Pirmasis metodas visados kintamąjį a priskirs reikšmei lygiai 0, nes reikšmė atkeliaus iš aukštesnių metodų ar kitų klasių ir niekados nepasieks antro ar trečio metodo, tačiau bus kreipiniai esant sąlygai. Tokios struktūros papildomas kodas nepareikalaus papildomų resursų, tačiau programišius gali aptikti kreipinius, kuriuos turės išanalizuoti, kai supras, kad programiniame kode yra nežinomas skaičius apsaugos metodų.

3. APSAUGOS METODO TAIKYMAS PROGRAMINĖJE ĮRANGOJE

Šioje dalyje aprašomi konkretūs žingsniai ką reikia daryti, norint integruoti siūlomą apsaugos metodą į programinę įrangą. Realizacijai bus naudojama programinis sprendimas, skirtas įmonei užsiimančiai prekyba. Programą sudaro pirkimo ir pardavimo dokumentai, ir su jais susijusi informacija, tokia kaip klientai, žaliavos ir sandėliai (3.1. Pav.).

18 Pav. Prekybos valdymo sistema

Programoje, kurioje bus realizuotas siūlomas apsaugos metodas, neturi jokių papildomų apsaugos priemonių, neturi nenaudojamo kodo, yra minimalios apimties tam, kad būtų:

- Realizuotas apsaugos metodas.
- Ištestuotas jo veikimas.
- Palyginta pasikeitusi kodo apimtis.
- Palygintas programos veikimo greitis.

Apsaugos metodo realizacijai bus naudojama:

- Microsoft Windows 8.1 operacinė sistema.
- Microsoft Visual Studio 2012 programavimo įrankis.
- C# programavimo kalba.
- „ConfuserEx“ kodo maskavimo įrankis.
- “ILSpy” įrankis skirtas sugeneruotą mašininį kodą atkurti į pirminį kodą .

3.1. Licencinio failo sukūrimas ir panaudojimas

Pirmiausia reikia sukurti licencinio failo prototipą, kadangi tai bus tik parodomasis failas, kurį panaudosim, ir jame esanti informacija gali priklausyti nuo to kas kuria tą failą. Realizacijoje naudojamas licencijos failas bus sudarytas XML failo struktūros pagrindu prieš šifravimą. Privalumas, jog lengvai galima struktūrizuoti tokio tipo duomenis naudojamoje programavimo kalboje, kas leidžia sukurti lentelės ar lentelių struktūrą. Failo struktūrą sudarys registracinio tipo duomenys:

- Asmeniniai unikalūs duomenys.
- Licencijos serijos numeris.
- Licencijos išrašymo ir galiojimo datos.

- Bitų sumos skaičiai.

Toliau pateikiamas tokio failo sukurtas atvaizdas (3.2. Pav.).

```
<?xml version="1.0" standalone="yes"?>
<content xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<Name>Donatas</Name>
<Surname>Kvietkus</Surname>
<LicenseNumber>A111B222C333</LicenseNumber>
<LicenseFromDate>2015-05-01</LicenseFromDate>
<LicenseToDate>2015-07-01</LicenseToDate>
<LicenceCode>7175A</LicenceCode>
</content>
```

3.2. Pav. Licencijos failas prieš šifravimą

Pateiktame atvaizde matomi laukų pavadinimai iš skaitinių reikšmių. Tai bitų sumos skaitinės vertės panaudojus du algoritmus. Pirmas ir paskutinis laukas saugoja tikrą bitų apimtį šešioliktainiame kode, o viduriniai skaičiai yra bitų apimtis, panaudojus maišos funkciją MD5 ir pasirinkus konkretų masyvo elementą. Žemiau pateikiamas dviejų algoritmų kodo fragmentas (3.3. Pav.).

```
private void CalculateCheckSum(string text)
{
    MD5 md5 = System.Security.Cryptography.MD5.Create();
    byte[] inputBytes = System.Text.Encoding.ASCII.GetBytes(text);
    byte[] hash = md5.ComputeHash(inputBytes);
    int checksum = 0;
    foreach (byte chData in inputBytes)
    {
        checksum += chData;
    }
    checksum &= 0xff;
    txtMD5.Text = hash[0].ToString();
    txtBits.Text = checksum.ToString("X2");
}
```

19 Pav. Bitų skaičiavimo algoritmas

Sukurtas licencijos failas turi būti užšifruotas. Šifravimui geriausia susikurti atskirą programą, kuri šifruos ir atšifruos failo duomenis. Taip galėsim patikrinti ar veikia šifravimas. Šifravimui naudojamas simetrinis šifravimo raktas, kaip šifruojamas ir atšifruojamas failas, yra pateikiamas supaprastintas kodo fragmentas (3.4. Pav.)

```
private void Encrypt(string text)
{
    byte[] password = GeneratePassword();
    byte[] inputBytes = System.Text.Encoding.ASCII.GetBytes(text);
    password = SHA256.Create().ComputeHash(password);
    inputBytes = CryptoClass.AES_Encrypt(inputBytes, password);
    System.IO.File.WriteAllText("Licence.lic",
        System.Text.Encoding.ASCII.GetString(inputBytes));
}
private string Decrypt()
{
    string text = System.IO.File.ReadAllText("Licence.lic");
    byte[] password = GeneratePassword();
    byte[] inputBytes = System.Text.Encoding.ASCII.GetBytes(text);
    password = SHA256.Create().ComputeHash(password);
    inputBytes = CryptoClass.AES_Decrypt(inputBytes, password);
    return System.Text.Encoding.ASCII.GetString(inputBytes);
}
```

20 Pav. Šifravimo ir atšifravimo kodo fragmentas

Pirmasis metodas „Encrypt“ šifruoja licencijoje naudojamą tekstą. Yra sukuriamas nekintantis slaptažodis, kuris dar perleidžiamas per maišos funkciją. Teksto šifravimui panaudojamas AES šifravimo algoritmas ir šifruotas tekstas įrašomas licencijos failą. Metodas „Decrypt“ skirtas šifruotą tekstą gražinti į pirminę padėti. Veiksmai vyksta analogiškai, kaip ir šifravimo metodo. Šifravimo metu naudojamas slaptažodis yra sukuriamas iš keturių skirtingų tekstinių simbolių eilučių, dėl priežasties, kad ilgi atviro teksto simboliai yra lengvai pastebimi, ir gali atkreipti programišiaus dėmesį. Panaudojus šifravimo algoritmą, gauname licencijos failą, kuriame jau nebesimato jokios failo struktūros, o tik įvairiausių simbolių aibė (3.5. Pav.)

```
W8ZFQQfzYbSfCr9XukafAraGLLPvfThua25LzJL67hsI1kSwLrTg0GVL/HvLJi3ab
5A2LoGJ0DwI3d8PgsGoHXbNDjGk1ggshkfCYDNuPoi03diHxZGPD6rh6ei0XUu0Wk
P9rMBQBNMJ+AplSmjx/ET0myPt
+30+Yo3Wx2rSvcL0p1k3aHwSkQahtBPLBHtn43Xb
+BhEP9rP2RNwNy9qVJEI9OcpYJ21cxJ4vMRrOwyz4Ofa/15u
+OWzEQHAC77dYDIgufT6CbkSnjKWNHaaB1Yr8MoxABHY2dyMme1812MI/iTCyXuRB
Lp2vMI37r7BJe0yWZxoaxWW78bMtCPmUXMFl6CIcQ6O7hCVXaZ1RIZomRC9n7GKnY
9Xq1jC3WWwOaHZweWgwycC1w5shwdtHMrnkF5ixgaBpSsWn56VJI
+KeSx1GHTntv1pV3kQ0UKx
```

21 Pav. Šifruotas licencijos failas

Sukūrus šifruotą licencijos failą, jo tikrinimo metodus reikia realizuoti pasirinktoje „Prekybos valdymo sistemoje“. Visų pirmą reikia panaudoti atšifravimo algoritmą, kuris jau buvo panaudotas šifravimo programoje (3.4. Pav.), tam, kad būtų galima nusiskaityti duomenis. Programoje yra klasė „Globals“, kurioje naudojami pirminio užkrovimo parametrai, todėl ten bus panaudotas tikrinimo metodas. Duomenų atšifravimui sukuriama atskira klasė. Tikrinimo metu bus lyginama licencijos data su kompiuterio data. Jei licencija bus galiojanti, programa bus paleidžiama darbui.

3.2. „Nematomo“ komponento sukūrimas ir panaudojimas

Viena iš lengviausių dalių yra sukurti komponentą ir jį kažkur tai panaudoti. Tačiau jį paslėpti yra sudėtingas uždavinys ir tam reikia atskirtų metodų. Kadangi naudojamos „Windows Forms“ nepalaiko nematomo komponento pelės žymeklio judėjimo, taip pat nepalaiko permatomų komponentų, bus imituojamas tokio komponento paslėpimas panaudojant visos formos žymeklio judėjimą. Tam, kad tai atlikti reikia įvykdyti šiuos žingsnius:

- Sukurti atsitiktinių reikšmių poziciją imituojančią komponentą bazinėje klasėje.
- Sukurti licencijos tikrinimą bazinėje klasėje.
- Naudojamoje klasėje aktyvuoti tikrinimą.

Metodas kuriamas bazinėje klasėje todėl, kad būtų jį galima iškviesti iš bet kurios paveldėtos klasės. Svarbu jog kvietimas nebūtų vykdomas kiekvienoje paveldėtoje klasėje, nes gali būti lengva suprasti kas vyksta, o šiuo atveju, tai gali atrodyti, jog programa tiesiog veikia nekorektiškai dėl programuotojo kaltės. Žemiau pateikiamas kodo fragmentas su atsitiktinių reikšmių parinkimo, nematomo mygtuko ir pelės žymeklio judėjimo metodais (3.6. Pav.).

```
protected void GenerateHiddenButton()
{
    button = new Button();
    button.Size = new Size(50, 50);
    button.Hide();
    button.Location = Position();
    this.Controls.Add(button);
}
private void baseForm_MouseMove(object sender, MouseEventArgs e)
{
    if (IsInControl(e.Location))
    {
        CheckLicence();
    }
}
private Point Position()
{
    int numberY = this.Size.Height;
    int numberX = this.Size.Width;
    Random random1 = new Random();
    Random random2 = new Random();
    int number1 = random1.Next(0, numberY);
    int number2 = random2.Next(0, numberX);
    Point point = new Point(number2, number1);
    return point;
}
```

22 Pav. Atsitiktinės pozicijos komponento kūrimas

Kiekvieną kartą, kai tik vyksta pelės žymeklio judėjimas, yra tikrinama, ar jo pozicija yra ties sukurtu komponentu. Buvo parinktas didesnis komponentas, tam, kad būtų galima testuoti jo veikimą. Licencijos tikrinimo dalyje yra nuskaitomas iš atminties, jau pradžioje užkrautas failas, ir panaudojus analogišką atšifravimo metodą. Atšifravimo metodas naudojamas kitas ,nes pradžioje naudotas metodas gali būti modifikuotas programišiaus.

Komponento licencijos tikrinimą sudaro:

- Failo iš atminties nuskaitymas.

- Teksto atšifravimas.
- Bitų sumos tikrinimas pagal numatytą struktūrą.

Pirmiausia, kad nuskaityti failą iš atminties, reikia ten jį patalpinti. Pradžioje, kai yra skaitomas licencijos failas pirminiame tikrinimo lygyje reikėtų panaudoti ir užkrovimą į atmintį. Vietoj pradžioje naudoto tiesioginio nuskaitymo, įdedame failo įkrovimą į atmintį (3.7. Pav.).

```
MemoryMappedFile MemoryMapped =
    MemoryMappedFile.CreateFromFile("helpers/Licence.lic",
    FileMode.Open, "License");
FileInfo fileInfo = new FileInfo("helpers/Licence.lic");
byte[] readBytes = new byte[fileInfo.Length];
using (MemoryMappedViewAccessor FileMap =
    MemoryMapped.CreateViewAccessor(0,fileInfo.Length))
{
    int i = 0;
    while (i < fileInfo.Length-1 )
    {
        readBytes[i] = FileMap.ReadByte(i);
        Console.WriteLine(FileMap.ReadByte(i));
        i++;
    }
}
string licenceText = System.Text.Encoding.UTF8.GetString(readBytes);
```

3.7. Pav. Failo nuskaitymas įkeliant į atmintį

Kai failas yra įkeltas į atmintį, jis turi kreipinio pavadinimą „License“, tokiu būdu galima kreiptis į atmintį jau vėlesniuose tikrinimuose ir nuskaityti joje esančia informaciją, kuri buvo licencijos faile. Tokiu būtų programa neturės daugiau tiesioginių kreipinių į licencijos failą, tai yra gerai, jei yra sekama kur vyksta kreipiniai į failą. Nuskaitytas tekstas toliau yra atšifruojamas ir paverčiamas į lentelės struktūros duomenis. Toliau vyksta tikrinimas su bitų skaičiavimu pagal numatytą struktūrą. Bitų suma yra apskaičiuojama sudėjus stulpeliuose esantį tekstą iki paskutinio stulpelio. Paskutiniame stulpelyje yra saugomas tos struktūros rezultatas, kuris ir bus lyginimas. Jei rezultatai nėra lygus, programa nustoja veikti.

3.3. Laiko intervalo fiksavimo ir nuskaitymo realizavimas

Laiko fiksavimas yra labai naudingas dalykas, jei kalbame apie programos riboto laiko galiojimą. Programa nereikalaujanti interneto prieigos, galinti dirbti vietiniame tinkle, ir laiko fiksavimas darbui įtakos neturi, gali būti įdiegta į virtualią mašiną, ar vietinį tinklą, kuriame laikas nuolatos atsukamas tam, kad galiotų licencija neribotą laiką. Papildoma apsaugos priemonė yra laiko fiksavimas reikšmės neturinčiuose failuose su paslėpto failo kopija. Tam, kad realizuoti tokį apsaugos metodą reikia atlikti šiuos žingsnius:

- Sukurti mažos reikšmės failą.
- Sukurti paslėptą failą naudojant algoritmą.
- Vykdyti failų skaitymą.
- Tikrinti fiksuojamų datų vientisumą.
- Įterpti fiksuojamą datą.

Mažos reikšmės failas gali būti paprasta informacija, kaip naudotis programa – vartotojo vadovas. Toks failas įprastai neturėtų užkliūti programišiui, kuris bandys programą naudoti nelegaliai. Failas

bus papildomas esama kompiuterio data (3.8. Pav.). Naudojamas trumpas datos formatas, tam, kad neapkrauti kompiuterio papildomu darbu, kuris tikrai ilgainiui didės dėl duomenų kaupimo. Duomenų kaupimas yra viena iš priežasčių, jog toks metodas ilgainiui bus pastebėtas, todėl reikėtų naudoti duomenų šalinimą kai failas pasiekia tam tikrą apimties ribą.

```
Sveiki, ačiū, kad naudojatės mūsų produktu, šiame dokumente bus pateikiama
trumpa informacija, kuri galbūt bus jums naudinga. Pirmiausia norint
naudotis mūsų programinia įranga turite užsiregistruoti mūsų svetainėje, ir
elektroniu paštu jums bus išsiųstas licencijos failas. Gautą licencijos
failą "License.lic" prašome įkelti į programos direktorija, kad galėtumėte
pradėti darbą. Įkėlus licencijos failą, du kartus pelės klavišu spustelėkite
"Programa.exe" failą. Jei licencija bus galiojanti, jūs būsite nukelti į
programos pradinį langą, ir galėsite pradėti naudotis mūsų programine
įranga.
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
;2015-05-01;2015-05-02;2015-05-03;2015-05-04;2015-05-06
```

3.8. Pav. Mažos reikšmės failas

Paslėptas failas sukuriamas atsižvelgiant į jo matomą variantą, tam, kad nesiskirtų struktūrą. Jis turės tokį pat pavadinimą, tik su papildomu identifikaciniu prierašu. Paslėpto failo kūrimo algoritmas pateikiamas žemiau (3.9. Pav.). Paslėpto failo kūrimui naudojama nestandartinė biblioteka[3]. Ji yra naudojama atviru kodu, todėl reikiamus metodus galima persikelti, ir neįtraukti bibliotekos į projektą, kad pavadinimas nesukeltų įtarimo.

```
private void CreateInvisibleFile()
{
    path = path + ":Hidden";
    if (!NtfsAlternateStream.Exists(path))
    {
        path = "Informacija.txt";
        string[] copyText = File.ReadAllLines(path);
        string allText = "";
        foreach (string item in copyText)
        {
            allText += item + System.Environment.NewLine;
        }
        path = path + ":Hidden";
        NtfsAlternateStream.WriteAllText(path, allText);
    }
}
```

3.9. Pav. Paslėpto failo kūrimas

Toliau yra sukuriami du atskiri metodai skirti nuskaityti sukurtus, matomą ir paslėptą failus. Nuskaitytų failų paskutinės eilutės, kuriuose yra užfiksuota data yra panaudojamos įvertinti datų vientisumą, tai reiškia, kad data fiksuota vakar, negali būti mažesnė nei fiksuota šiandien, ar rytoj. Mažos reikšmės faile (3.8. Pav.) yra surašytos iš eilės einančios datos, jei paskutinė data bus mažesnė

[3] *CodeFluent.Runtime.BinaryServices*

nei priešpaskutinė, programa nustos veikti. Tiek matomame, tiek paslėptame failuose vyksta analogiškas datų vientisumo patikrinimas (3.10. Pav.).

```
private bool IsDateIntegrity(string[] dates)
{
    for (int i = 1; i < dates.Length; i++)
    {
        if(Convert.ToDateTime(dates[i]) < Convert.ToDateTime(dates[i - 1]))
        {
            return false;
        }
    }
    return true;
}
```

23. Pav. Datų vientisumo tikrinimas

Abu tikrinimo metodai yra paleidžiami panaudojant gijas, tam, kad būtų vykdomi vienu metu. Programos darbas tikrinimo metu nuo to tik pagreitės, ir nesukels papildomų įtarimų, o lygiagretūs kreipiniai į failus neleis taip lengvai aptikti paslėpto failo.

3.4. Nenaudingo kodo įterpimas

Dažniausiai visos programinės įrangos, kurios pasiekia vartotoją, turi savyje daug nenaudojamo kodo, kuris užima vietą ir neatlieka nieko naudingo. Tačiau yra atvejų, kai nenaudojamas kodas yra naudingas, ir gali padėti apsaugoti programinę įrangą. Toks kodas visų pirmą turi turėti kreipinius, nors tie kreipiniai niekados ir nepasieks tos vietos. Antra, tai programa turi turėti apsaugos mechanizmą, kurį bus sunku aptikti būtent dėl nenaudingo kodo panaudojimo. Toliau yra pateikiamas kodo fragmentas, kuris buvo panaudotas (3.11. Pav.).

```
public bool IsNeedActivate(int a)
{
    int b = 0;
    if (a + b == 0)
    {
        return true;
    }
    else if (a != 0)
    {
        CheckDate();
        if (b == 0)
        {
            Activate();
            return true;
        }
        return false;
    }
    else if (IsAdmin())
    {
        return true;
    }
    return false;
}
```

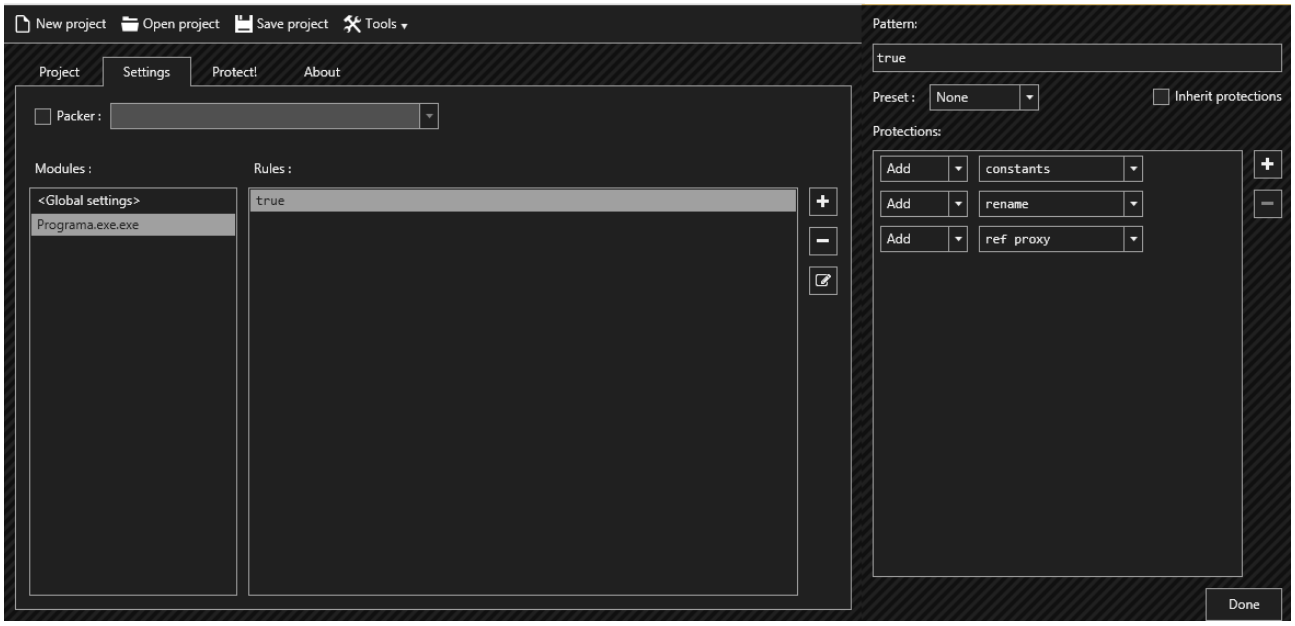
24 Pav. Nenaudingas kodas

Kadangi metodas yra viešas jis naudojamas keliuose klasėse, o paduodama reikšmė visados pasiekią pirmą sąlygos dalį. Kiti metodai yra nenaudojami, tačiau turi kreipinius, o tuose metoduose yra kiti metodai, taip sudaroma medžio struktūra. Aptikus saugos metodus, susidaro įspūdis jog jų yra didelis kiekis, todėl programišiui gali tekti išanalizuoti visą programinį kodą. Jei kodas užmaskuotas, tai padaryti tampa dar sudėtingiau.

3.5. Kodo maskavimas panaudojant jau sukurtus maskavimo įrankius

Analizės dalyje buvo aprašytas keletas kodo maskavimo įrankių, tačiau tik vienas iš jų, kuris suveikė, yra nemokamas, galima naudoti neribotą laiką, ir keisti programinį kodą pagal save. Kai programa jau yra pilnai paruošta būti pateikta vartotojui, paskutinis žingsnis, kurį reikia atlikti, tai užmaskuoti kodą. Tam panaudosime „ConfuserEx“ įrankį, kuris turi visus standartinius apsaugos metodus. Apsaugai bus panaudotas tik metodų, klasių ir nekintamų dydžių pervardinimas (3.12. Pav.). Pasirinkto kodo maskavimo įrankio veikimo principas yra paprastas, įsikeliame norimą sugeneruotą programinį kodą su plėtinių „*.exe“[4], pasirenkame kokias taisykles pridėsime ir galima vykdyti kodo maskavimą.

[4] * - bet koks pavadinimas



25 Pav. "ConfuserEx" maskavimo įrankis

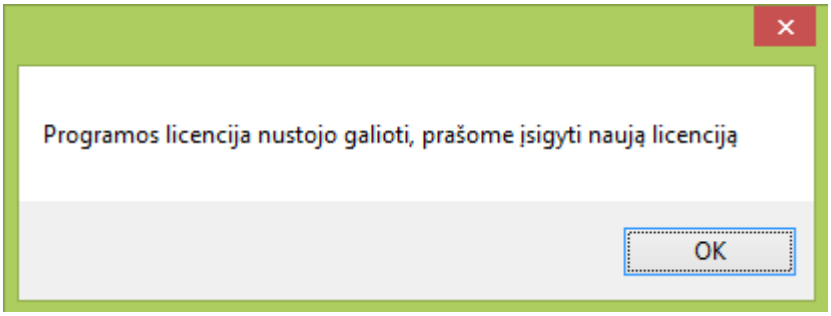
Atlikus kodo maskavimą, programinė įranga yra pilnai paruošta veikti pagal pasiūlytą dviejų lygių apsaugos modelį.

3.6. Programos veikimas

Atlikus visus punktus, patikrinamas programos veikimas. Tam, kad patikrinti, kaip veikia ir apsaugos metodai, reikia susikurti du papildomus licencijos failus:

- Pasibaigusio galiojimo.
- Modifikuotą.

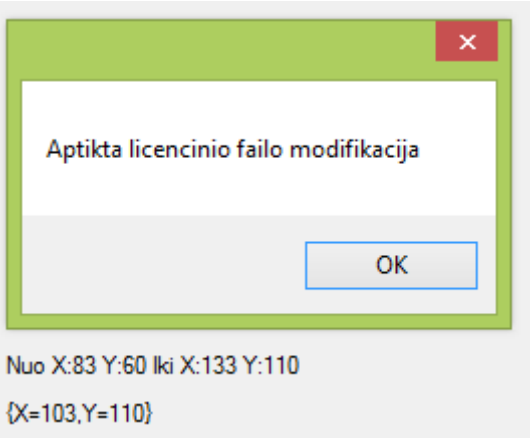
Jų pagalba bus nustatoma ar programa funkcionuoja tinkamai, ar apsaugos metodai suveikia. Pirmiausia įkeliamas pasibaigusios licencijos failas. Programos pajungimo metu iš karto gaunamas pranešimas, kuris praneša jog licencija yra nebegaliojanti (3.13. Pav.).



26. Pav. Pasibaigusios licencijos rezultatas

Gavus tokį pranešimą, aišku jog pirmoji apsauga suveikė, ir programos nepavyks įjungti, jei licencijos failas bus su pasibaigusia galiojimo data, arba jei nebus panaikintas licencinio failo reikalavimas panaudojant specialius įrankius derinant kodą mašininio kodo lygmenyje. Sekantis žingsnis įkelti modifikuotą licencijos failą. Tai failas, kuriame yra pakeista data, tačiau nėra pakeistas bitų sumos skaičiavimo rezultatas. Įkėlus modifikuotą licencijos failą, ir įjungus langą „Naujas pirkimo

dokumentas“, kuriame ir yra paslėptas komponentas, po kurio laiko slenkant pelės žymekliu po programos langą, programa nustos veikti. Užfiksuoti, kaip programa nustoja veikti nėra galimybės, todėl buvo išvedamas pranešimas (3.14. Pav.).



27. Pav. Modifikuoto licencinio failo rezultatas

Taip pat po pranešimu yra rodomos koordinatės, ties kuriomis įvyko patikrinimas, kad sužinoti kurioje vietoje buvo sugeneruotas paslėptas mygtukas.
Kitam apsaugos metodui patikrinti, reikia modifikuoti failą, kuris yra naudojamas datos fiksavimui. Failo „Informacija.txt“ duomenis pakeitus į tokius, kurie atsirastų, po datos atsukinėjimo kompiuteryje, ir įjungus langą „Naujas pardavimo dokumentas“, iškart bus gautas analogiškas pranešimas, kaip, kad su modifikuota licencija.

4. IŠVADOS

Šiuo metu didelę problemą sudaro programinės įrangos apsaugojimas, nuo nelegalaus jos panaudojimo. Atlikus analizę paaiškėjo jog esami metodai yra efektyvus, tačiau nepakankamai, nes kiekvienais metais didėjantys kaštai apsaugojant programinę įrangą rodo, kad reikia keisti strategija.

Siūlomas sprendimas apgauti programišių, sukuriant iliuzinę apsaugą, yra vienas iš būdų, kaip laimėti daugiau laiko. Jei laikas tampa pakankamai didelis, nemaža dalis vartotojų pagalvos, ar verta naudoti nekokybiškai veikiančia nemokama, ar kokybiškai veikiančia licencijuota programine įranga.

Buvo sukurtas projektinis apsaugos metodo modelis, kuriame aprašoma, kaip reikėtų elgtis, ir ką reikėtų naudoti, norint apsaugoti programinę įrangą galimai ilgesnį laiką. Tiek modelis, tiek metodas, nėra visapusiška apsauga nuo nelegalaus programinės įrangos vartojimo.

Aprašytas apsaugos modelis buvo pritaikytas programinėje įrangoje. Realizuota dviejų lygių apsauga, kur pirmasis lygis skirtas fiktyviai apsaugai, kurią įveikus programa pradeda veikti pilnu rėžimu. Antro lygio apsauga suveikia besinaudojant programine įranga ir aptikus nelegalų naudojimą – nustoja veikusi. Buvo ištestuotas veikimas rankiniu būdu, programa veikia kaip ir buvo numatyta projekto kūrimo metu. Programa yra lengvai pažeidžiama, jei yra žinomi apsaugos metodai, todėl tai įrodo, kodėl dauguma programinės įrangos kūrėjų neatskleidžia savo programinės įrangos apsaugos sprendimų.

5. LITERATŪRA

1. Alliance, B. |. (2015 m. 05 20 d.). *Security Threats Rank as Top Reason Not to Use Unlicensed Software*. Nuskaityta iš http://globalstudy.bsa.org/: http://globalstudy.bsa.org/2013/
2. babelfor.NET. (2015 m. 05 20 d.). *Obfuscation and Licensing for .NET*. Nuskaityta iš https://babelfor.net.
3. Bali, P. (n.d.). COMPARATIVE STUDY OF PRIVATE AND PUBLIC KEY CRYPTOGRAPHY ALGORITHMS: A SURVEY.
4. *Boomerang decompiler*. (2015 m. 05 20 d.). Nuskaityta iš http://sourceforge.net/: http://sourceforge.net/projects/boomerang/
5. BV, T. S. (2015 m. 05 20 d.). *TIOBE Index for May 2015*. Nuskaityta iš http://www.tiobe.com/: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html
6. Christian Collberg, C. T. (1999). *Software Watermarking: Models and Dynamic Embeddings.* San Antonio Texas USA .
7. Christian Collberg, G. M. (2003). Sandmark - a tool for software protection reasearch.
8. Christian Collberg, J. N. (2009). *Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection.*
9. Coe, M. G. (2015 m. 05 20 d.). *Modern Cryptography - Methods and Uses*. Nuskaityta iš http://h2g2.com/: http://h2g2.com/approved_entry/A1315919#conversations
10. Decryption, E. A. (2015 m. 05 20 d.). *Encryption Algorithms*. Nuskaityta iš http://www.encryptionanddecryption.com/: http://www.encryptionanddecryption.com/algorithms/encryption_algorithms.html
11. Duran, P. F. (2015 m. 05 11 d.). *http://blog.codefluententities.com/*. Nuskaityta iš http://blog.codefluententities.com/2013/03/14/manipulating-ntfs-alternate-data-streams-in-c-with-the-codefluent-runtime-client/
12. Eilam, E. (2005). *Reversing: Secrets of Reverse Engineering.*
13. Foundation, F. S. (2015 m. 05 20 d.). Nuskaityta iš https://yck1509.github.io: https://yck1509.github.io/ConfuserEx/
14. Halina Kaminski, M. P. (n.d.). The Pattern Language of Software Licensing.
15. Hernandez, P. (2015 m. 05 20 d.). *Microsoft: Pirated Software to Cost Businesses $491 Billion in 2014*. Nuskaityta iš http://www.eweek.com/: http://www.eweek.com/security/microsoft-pirated-software-to-cost-businesses-491-billion-in-2014.html
16. Yadav, A. (2015 m. 05 20 d.). *C# Multithreading Example*. Nuskaityta iš http://resources.infosecinstitute.com/: http://resources.infosecinstitute.com/multithreading/

17. ICSharpCode. (2015 m. 05 20 d.). *ILSpy*. Nuskaityta iš http://ilspy.net/.

18. Irvine, K. (2006). *Assembly Language for Intel-Based Computers (5th Edition).* Upper Saddle River, NJ, USA.

19. Jasvir Nagra, C. T. (2001). Software Watermarking: Protective Terminology.

20. Khandelwal, S. (2015 m. 05 20 d.). *Windows? NO, Linux and Mac OS X Most Vulnerable Operating System In 2014*. Nuskaityta iš www.http://thehackernews.com/: http://thehackernews.com/2015/02/vulnerable-operating-system.html

21. LeXtudio. (2015 m. 05 20 d.). *Obfuscar Open source obfuscator for .NET.* Nuskaityta iš http://lextudio.com/: https://obfuscar.codeplex.com/

22. *List of obfuscators for .NET*. (2015 m. 05 20 d.). Nuskaityta iš http://en.m.wikipedia.org: http://en.m.wikipedia.org/wiki/List_of_obfuscators_for_.NET

23. M. Friedl, N. P. (2015 m. 05 20 d.). *Diffie-Hellman Group Exchange for the Secure Shell (SSH) Transport Layer Protocol*. Nuskaityta iš http://www.ietf.org/: http://www.ietf.org/rfc/rfc4419.txt

24. maldevel. (2015 m. 05 20 d.). *POPULAR SYMMETRIC ENCRYPTION ALGORITHMS*. Nuskaityta iš http://securityblog.gr/: http://securityblog.gr/431/popular-symmetric-encryption-algorithms/

25. Mansoor Ebrahim, S. K. (2013). *Symmetric Algorithm Survey: A Comparative Analysis.*

26. McDonald, N. G. (2010). PAST, PRESENT, AND FUTURE METHODS OF CRYPTOGRAPHY AND DATA ENCRYPTION.

27. Microsoft. (2015 m. 05 20 d.). *Files and Clusters*. Nuskaityta iš https://msdn.microsoft.com: https://msdn.microsoft.com/en-us/library/aa364056.aspx

28. Myles, G. (2004). *Using software watermarking to discourage piracy.* New York, NY, USA .

29. Mohab U. AbdelHameed, M. A. (2009). Portable Executable Automatic Protection using Dynamic Infection and Code Redirection.

30. Poulsen, K. (2015 m. 05 20 d.). *Microsoft: Closed source is more secure*. Nuskaityta iš http://www.securityfocus.com/: http://www.securityfocus.com/news/191

31. *Public-key cryptography*. (2015 m. 05 20 d.). Nuskaityta iš http://en.wikipedia.org: http://en.wikipedia.org/wiki/Public-key_cryptography

32. Rouse, M. (2015 m. 05 20 d.). *Reverse Engineering Definition*. Nuskaityta iš http://searchsoftwarequality.techtarget.com/: http://searchsoftwarequality.techtarget.com/definition/reverse-engineering

33. Rouse, M. (2015 m. 05 20 d.). *Reverse Engineering Definition*. Nuskaityta iš http://searchsoftwarequality.techtarget.com/: http://searchsoftwarequality.techtarget.com/definition/reverse-engineering

34. RUSTEMSOFT. (2015 m. 05 20 d.). *Skater .NET Obfuscator*. Nuskaityta iš http://rustemsoft.com/: http://rustemsoft.com/obfuscator.asp

35. Services, T. (2015 m. 05 20 d.). *Software License Types*. Nuskaityta iš https://tulane.edu: https://tulane.edu/tsweb/software/software-license-types.cfm

36. Shetty, S. (n.d.). A Review on Asymmetric Cryptography – RSA and ElGamal Algorithm.

37. Software, L. (2015 m. 05 20 d.). Nuskaityta iš http://www.ssware.com/.

38. Software, L. (2015 m. 05 20 d.). *5 Reasons You Should Buy Crypto Obfuscator Now* . Nuskaityta iš http://www.ssware.com/: http://www.ssware.com/cryptoobfuscator/order.htm

39. Vaughan-Nichols, S. J. (2015 m. 03 27 d.). *The most popular US end-user operating systems, according to the federal government*. Nuskaityta iš http://www.zdnet.com/: http://www.zdnet.com/article/the-federal-government-on-what-are-the-most-popular-us-end-user-operating-systems/

40. William Zhu, C. T. (2006). Recognition in Software Watermarking. *Proceedings of the 4th ACM international workshop on Contents protection and security*, 29 - 36.