VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ
DEPARTMENT OF COMPUTER SYSTEMS

DIGITÁLNÍ STEGANOGRAFIE PRO SPUSTITELNÉ SOUBORY

DIGITAL STEGANOGRAPHY FOR EXECUTABLES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ĽUBOŠ BEVER**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JOSEF STRNADEL, Ph.D.**
SUPERVISOR

BRNO 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačových systémů (UPSY) Akademický rok 2021/2022

Zadání bakalářské práce

Student: **Bever Ľuboš**
Program: Informační technologie
Název: **Digitální steganografie pro spustitelné soubory**
Digital Steganography for Executables
Kategorie: Bezpečnost

Zadání:

1. Vytvořte přehled metod z oblasti digitální steganografie, zúženěji a detailněji pak přehled z podoblasti steganografie pro skrývání informace ve spustitelných souborech (dále jen "steganografie"), jejich vlastností a shrňte současný stav a trendy v této zúžené podoblasti.
2. Zvolte typ skrývané informace (textová, obrazová apod.) a její vlastnosti. Na základě existující či vlastní analýzy formátů spustitelných souborů a typu skrývané informace zvolte vhodné formáty spustitelných souborů a vhodné steganografické metody.
3. V souladu s bodem 2 připravte vhodnou sadu dat pro ověřování vlastností zvolených steganografických metod a navrhněte mechanismus vyhodnocování jejich vlastností na základě těchto dat.
4. Implementujte několik existujících metod steganografie pro spustitelné soubory, zvažte jejich modifikace, popř. návrh a implementaci vlastních metod.
5. Vhodně ověřte funkčnost implementovaných metod; vyhodnoťte jejich vlastnosti a porovnejte je jak navzájem, tak s daty z několika publikací.
6. Shrňte dosažené výsledky, diskutujte možné směry využití a rozvoje předloženého řešení.

Literatura:

- Dle pokynů vedoucího.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Strnadel Josef, Ing., Ph.D.**
Vedoucí ústavu: Sekanina Lukáš, prof. Ing., Ph.D.
Datum zadání: 1. listopadu 2021
Datum odevzdání: 11. května 2022
Datum schválení: 29. října 2021

Abstrakt

Steganografia pre spustiteľné súbory je najmenej rozšírenou steganografiou. Výskumy z tejto oblasti utíchli po niekoľkých, len málo, pokusoch o jej implementáciu. Cieľom práce je implementácia existujúcich metód a návrh ich modifikácie. Týmto vznikol rozšíriteľný software, ktorý je možné použiť pre implementáciu ďalších metód. Implementované metódy boli riadnym testovaním zhodnotené a porovnané. Výsledky porovnania ukazujú, že použitá metóda substitúcie inštrukcií približne odpovedá jej referenčnej hodnote $\frac{1}{110}$, avšak výsledky sú veľmi závislé od vstupných binárnych súborov. Navrhnuté rozšírenie tejto metódy dosahuje v priemere dátovú rýchlosť $\frac{1}{84}$, čo je len o $1,5$-krát menej ako hodnota získaná z inej existujúcej implementácie, v ktorej bol pre hľadanie ekvivalenčných tried použitý špecializovaný software. Maximálna dátová rýchlosť získaná z testovacích programov je $\frac{1}{38}$.

Abstract

Steganography for executable files is the least common steganography. Research in this area has subsided after several, not many, attempts to implement it. The aim of this work is the implementation of existing methods and its modification proposal. Extensible software that has been created, can be also used to implement other methods. The implemented methods were properly tested, evaluated and compared. The comparison results show, that the used instruction substitution method, roughly corresponds to its reference value $\frac{1}{110}$, however the results are highly dependent on the input binaries. The proposed extension of this method averages a data rate of $\frac{1}{84}$, which is only 1.5 times less than the value obtained from another existing implementation in which specialized software was used to search for equivalence classes. The maximum data rate obtained from test programs is $\frac{1}{38}$.

Kľúčové slová

digitálna steganografia, spustiteľný súbor, formát ELF, formát PE, inštrukčná sada x86-64, AMD64, Intel 64, substitúcia inštrukcií, inštrukcie NOP, vkladanie informácií, extrakcia informácií

Keywords

digital steganography, executable file, ELF format, PE format, instruction set x86-64, AMD64, Intel 64, instruction substitution, NOP instructions, information embedding, information extraction

Citácia

BEVER, Ľuboš. *Digitální steganografie pro spustitelné soubory.* Brno, 2022. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Josef Strnadel, Ph.D.

Digitální steganografie pro spustitelné soubory

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Josefa Strnadela, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Ľuboš Bever
10. mája 2022

Poďakovanie

Chcem sa poďakovať svojmu vedúcemu práce Ing. Josefovi Strnadelovi, Ph.D. za odborné rady, cenné pripomienky, trpezlivosť, ochotu a pomoc, ktorú mi poskytol pri písaní bakalárskej práce. Moja vďaka tiež patrí rodine a priateľke za podporu nielen pri písaní tejto práce.

Obsah

1 Úvod **3**

2 Skrývanie dát v rámci iných dát **5**
2.1 Úvod do digitálnej steganografie 5
2.2 Skrývanie dát v texte 12
2.3 Skrývanie dát v obrázkoch 13
2.4 Skrývanie dát vo zvuku 13
2.5 Skrývanie dát vo videu 14
2.6 Skrývanie dát v internetovej sieti 15
2.7 Iné možnosti ukrytia dát 15

3 Digitálna steganografia spustiteľných súborov a ich analýza **16**
3.1 Analýza formátu ELF 16
3.2 Analýza formátu PE 20
3.3 Spôsoby ukrytia dát v spustiteľných súboroch 25
3.4 Existujúce programové vybavenie 28

4 Návrh steganografického nástroja **30**
4.1 Architektúra aplikácie 30
4.2 Popis zvolenej steganografickej metódy 33
4.3 Popis rozšírenia použitej steganografickej metódy 35
4.4 Teoretický návrh ďalších nájdených redundancií 36
4.5 Vyhodnocovanie vlastností steganografických metód 38
4.6 Použité technológie 38

5 Implementácia steganografického nástroja pre spustiteľné súbory **39**
5.1 Spustenie steganografického nástroja 39
5.2 Disassembling 40
5.3 Príprava potrebných dát pred analýzou inštrukcií 41
5.4 Selekcia a analýza inštrukcií 42
5.5 Predspracovanie dát a vkladanie 43
5.6 Proces extrakcie a následné spracovanie dát 45

6 Testovanie a experimenty nad implementovanými metódami **46**
6.1 Testovanie nepostrehnuteľnosti metód 46
6.2 Porovnanie dátovej rýchlosti jednotlivých metód 47
6.3 Porovnanie časovej zložitosti a kapacity 48
6.4 Porovnanie kapacity a veľkosti súborov 49

7 Záver **51**

Literatúra **52**

A Obsah priloženého pamäťového média **58**

B Špecifikácia metód jednotlivých digitálnych steganografií **59**
B.1 Textová steganografia 59
B.2 Obrazová steganografia 62
B.3 Zvuková steganografia 65
B.4 Videosteganografia 68

Kapitola 1

Úvod

Ukladanie a výmena súkromných informácií je základnou aktivitou všetkých používateľov internetu. Informácie sú zdrojom vedomostí, preto je nutné ich chrániť. Jednou z praktík informačnej bezpečnosti je steganografia. Steganografia, alebo skrývanie informácií, predstavuje významnú hrozbu pre vládne inštitúcie či spoločnosti a ich digitálne produkty. Existujú dokonca záznamy, že bola použitá pri plánovaní teroristického útoku na Svetové obchodné centrum v New Yorku z roku 2001.

Neoprávnený prístup k dôverným informáciám je v súčasnom digitálnom svete veľkým lákadlom pre útočníkov. Existujú tak dôvody, kvôli ktorým vznikajú rôzne snahy a opatrenia proti takýmto činom. Je preto zrejmé, že koncept steganografie majú v záujme rozvíjať obe strany tejto situácie. Keďže veľkosť multimediálnych objektov je pre ľudské porozumenie enormná, bolo nájdených veľa spôsobov, ako steganografiu pomerne bezpečne použiť. Skutočnosťou však zostáva, že s príchodom nových metód prichádzajú aj techniky, ktoré ich bezpečnosť ochromujú. Ide o nekonečný cyklus podobný kryptografii a kryptoanalýze.

Skrývanie dát v rámci spustiteľných súborov tvorí osobitnú kapitolu celej problematiky steganografie. Ide o podobnú praktiku vkladania dát do programov, ako je tomu pri injekcii škodlivých častí kódu. Nielenže zámena jediného bitu informácie môže znefunkčniť celý program, ale je nutné myslieť aj na detekciu tajných dát antivírusovými programami. Tie sú dnes schopné detegovať rôzne, dokonca aj predtým nimi nepoznané, anomálie. Ide o faktory znevýhodňujúce skrývanie dát v rámci spustiteľných súborov, a preto doposiaľ v tomto uplynulý výskum je v porovnaní s ostatnými druhmi steganografie viditeľne menší.

Existuje pomerne malé množstvo techník, ktorými je možné ukryť dáta v programoch. Navyše, tieto techniky sú často významne závislé od inštrukčnej sady procesorov a sú založené na hĺbkovej analýze formátov spustiteľných súborov. Aplikácia steganografických techník spustiteľných súborov je často veľmi zložitá, pričom poskytuje len malú kapacitu. Výskumy doteraz ukazujú, že ak sa steganografia pre spustiteľné súbory chce vyrovnať inej steganografii (napr. obrazovej), je potrebné jej techniky kombinovať.

Experimentálne zhodnotenie techník a ich použiteľnosti je dôležitým konceptom pri návrhu nových metód akejkoľvek steganografie. Preto časť náplne tejto práce tvorí skúmanie súčasných praktík takéhoto zhodnotenia a použitie čo najúčinnejšieho mechanizmu. Témou práce je implementácia rozšíriteľného software, ktorý implementuje niektoré existujúce steganografické metódy pre spustiteľné súbory a experimentálne použitie mechanizmu pre porovnanie týchto metód a zhodnotenie ich vlastností.

Na začiatok, kapitola 2 poskytuje čitateľovi komplexný prehľad digitálnej steganografie, ktorý si je možné dodatočne rozšíriť prílohou B. Obsahom kapitoly 3 je analýza formátov spustiteľných súborov, s ktorými táto práca počíta, predstavenie súčasných metód stegano-

grafie spustiteľných súborov a prehľad aktuálne dostupných programových vybavení v tejto oblasti. Cieľom kapitoly 4 je návrh programu, ktorý sa týka najmä použitých metód a prevedeného hľadania ďalších možností steganografie, a idea vyhodnotenia metrík, ktorým sa budú jednotlivé metódy testovať a porovnávať. Plynule nadväzuje kapitola 5, ktorá popisuje implementáciu navrhnutého programu. Práca pokračuje popisom uskutočneného testovania a experimentovania s implementovanou aplikáciou. Na záver sa v kapitole 7 zhodnotí dosiahnutý cieľ, získané výsledky a navrhnú sa možné vylepšenia.

Kapitola 2

Skrývanie dát v rámci iných dát

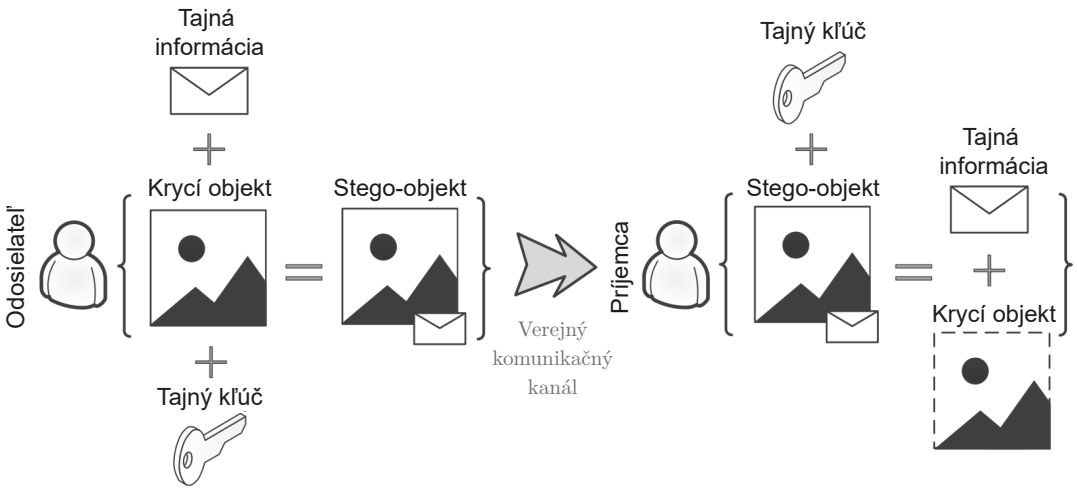
V súčasnosti sa digitálna steganografia delí na niekoľko druhov podľa typu digitálneho objektu, v rámci ktorého sú tajné dáta skrývané. Podkapitola 2.1 priblíži digitálnu steganografiu a pojmy s ňou súvisiace. Zvyšok tejto kapitoly predstavuje jednotlivé druhy steganografie a vytvára prehľad ich metód. Prvým z nich je skrývanie dát v texte (podkapitola 2.2), neskôr v obrázkoch (podkapitola 2.3), vo zvuku (podkapitola 2.4), vo videu (podkapitola 2.5), v sieti (podkapitola 2.6) a nakoniec sú spomenuté ďalšie zaujímavé druhy steganografie (podkapitola 2.7), ktoré naznačujú, kam aktuálny vedecký výskum v tejto oblasti smeruje. V tejto kapitole je vynechaná steganografia spustiteľných súborov, ktorej sa užšie, no detailnejšie venuje nasledujúca kapitola 3.

2.1 Úvod do digitálnej steganografie

V dnešnom digitálnom svete sa informácie prenášajú internetom častejšie ako kedykoľvek predtým. Preto existencia technológie zabezpečenia a ochrany citlivých a súkromných správ je takmer nevyhnutná. *Digitálna steganografia* (ďalej len *steganografia*) je umenie a veda nenápadnosti ukrytia informácií v skrytých kanáloch, aby sa zabránilo ich odhaleniu. Pojem steganografia pochádza z gréčtiny, kde *steganos* znamená „zakryté“ alebo „skryté“ a *graphein* znamená „písať“. Jej cieľom je skryť informáciu v rámci nosného digitálneho objektu (skrytý kanál) tak, aby nikto okrem odosielateľa a príjemcu netušil o prítomnosti skrytej informácie. [7] [39] [23]

Steganograf je osoba, ktorá aplikuje steganografickú metódu. Digitálne objekty v rámci steganografického systému obsahujúce skrytú informáciu sa nazývajú *stego-objekty* (niekedy aj *steganogramy*) a objekty, ktoré ju neobsahujú sa nazývajú *krycie objekty*. Vďaka tomu, že sú tajné informácie skrývané vo vnútri multimediálnych objektov (text, obrázok, sieťové pakety, . . .), môžu byť prenášané otvoreným komunikačným kanálom, keďže vedomosť o existencii skrytej informácie má len odosielateľ a príjemca. Informácie samotné, ako aj krycí objekt, ktorý ich skrýva, môžu byť rozdielnymi digitálnymi objektmi. [7] [43] [17]

Niekedy sa používa aj *steganografický kľúč*, ktorý je tajný. Tento kľúč riadi proces vkladania a extrakcie informácie. Jeho úlohou môže byť napr. rozptýlenie tajnej informácie na podmnožinu všetkých vhodných miest v krycom objekte. Bez kľúča je táto podmnožina neznáma, a teda nie je možné informáciu spätne reprodukovať/extrahovať. Útočník by sa, pri snažení nájsť ukrytú informáciu, dostal len k zmesi použitých a nepoužitých miest v krycom objekte. Obrázok 2.1 znázorňuje steganografický systém, ktorý používa takýto kľúč. [47] [43] [26]



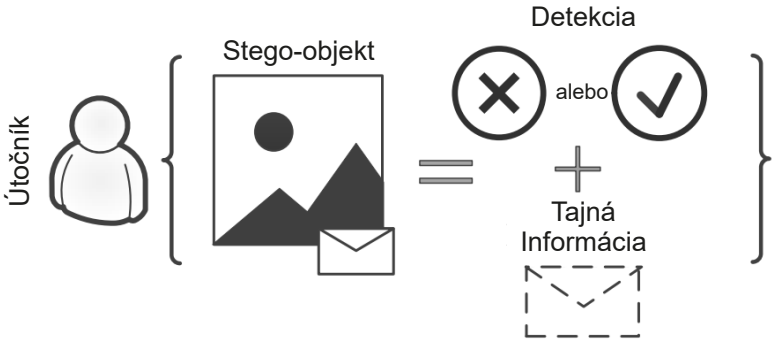
Obr. 2.1: **Proces digitálnej steganografie** – vkladanie a extrakcia tajnej informácie za použitia steganografického kľúča. (Prevzaté a preložené z [17])

Osobitnú pozornosť si zaslúži samotný úvod tejto práce (kapitola 1). Výberom vždy prvého písmena každého odseku vzíde tajná správa v anglickom jazyku: „UNSEEN“. V prípade, že až do tohto momentu čitateľ práce nedostal podozrenie, že tento text by mohol byť stego-objektom, potom sa pokus o použitie jednoduchej steganografie vydaril. Keďže steganografia ako taká počíta s ľudskou naivitou, je relatívne vysoká pravdepodobnosť, že stego-objekt bol detegovaný až v tejto chvíli. Avšak pokiaľ prvý odsek úvodu práce spôsobil akékoľvek pochybenie u čitateľa, snaha o použitie steganografickej metódy sa nepodarila (príklad inej nevhodnej steganografickej metódy znázorňuje obrázok 2.2). Od tejto chvíle je táto práca odhalená ako nosný objekt steganografie, a teda pokus o ukrytie ďalšej tajnej správy v rámci tohto textu by bol riskantný. [7]

```
randoM capitalosis is a rarE disEase ofTen
contrAcTed by careless inTernet users. tHis sad
illnEss causes the aFfected peRsON To randomly
capitalize letters in a bOdy oF texT. please
do not confuse this disease witH a blatant
attEmpt aT steganogRAPhy.
```

Obr. 2.2: **Príklad zlej steganografickej metódy** – skrytá informácia je ľahko pozorovateľná napriek tomu, že je asi desaťkrát menšia ako krycí objekt. (Prevzaté z [7])

Steganalýza (obrázok 2.3) sa oproti tomu snaží objaviť prítomnosť skrytých informácií. *Steganalytické systémy* sa používajú na zistenie, či digitálny objekt obsahuje tajnú správu. Ide o veľmi náročnú disciplínu, keďže jej úspech je založený na zraniteľnostiach steganografických techník. [15] [39]



Obr. 2.3: **Proces steganalýzy** (Prevzaté a preložené z [17])

2.1.1 Vlastnosti digitálnej steganografie

Pre steganografiu sú definované tri vlastnosti, vďaka ktorým je vhodná na skrývanie informácií: [5] [27] [15]

1. *Nepostrehnuteľnosť* – môže ísť o číslo určujúce kvalitu stego-objektu, ako maximálny pomer signálu k šumu, kde vyššie číslo implikuje vyššiu kvalitu stego-objektu. Ide o naplnenie základnej požiadavky steganografie, a to aby stego-objekt bol ľudským vnemom nerozoznateľný od krycieho objektu tak, aby nevzbudzoval podozrenie. Teda tajná správa musí spôsobiť len nepatrné zmeny krycieho objektu. V prípade odlišných krycích objektov môže byť táto vlastnosť definovaná rôzne.

2. *Robustnosť (odolnosť)* – popisuje odolnosť skrytej informácie voči zmenám krycieho objektu (pridanie náhodného šumu, stratová kompresia, ...). Môže byť vyjadrená číslom, ktoré vznikne podielom kvality neporušeného stego-objektu ku kvalite stego-objektu narušeným steganalytickými zásahmi.

3. *Kapacita* – číslo určujúce maximálny počet bitov tajnej informácie, ktoré je možné ukryť v rámci krycieho objektu.

Literatúra niekedy odčleňuje od vlastnosti nepostrehnuteľnosť ďalšiu vlastnosť – *nedetegovateľnosť*. V tomto prípade ide o odolnosť steganografickej metódy voči detekcii jej použitia za pomoci steganalytických prístupov (napr. štatistické techniky atď.). [18]

2.1.2 Bezpečnosť digitálnej steganografie

Vedecké štúdium steganografie odštartoval Gustavus J. Simmons v roku 1983, kedy predstavil klasický steganografický model hovoriaci o plánovaní úteku dvoch väzňov. Tí si vymieňajú správy kontrolované dozorcom, a preto musia tajiť svoje plány v rámci neškodne pôsobiacich (krycích) objektov. Týmto si sú schopní navzájom vymeniť svoje stego-objekty. Tie sú posielané verejným kanálom a dozorca tak môže svojvoľne kontrolovať všetky ich správy. Dozorca môže ku kontrole pristúpiť nasledovne: [39] [3] [21]

- **Aktívne** – dozorca zakaždým pozmení správu od oboch väzňov, aj keď nemusí mať podozrenie, že ide o stego-objekt.

- **Pasívne** – dozorca kontroluje všetky správy a snaží sa zistiť, či ide o stego-objekt, pričom v prípade podozrenia zasiahne.

V digitálnom svete by príkladom aktívneho prístupu dozorcu mohlo byť pozmenenie správy v podobe rôznych operácií nad vymieňajúcim objektom. Stratová kompresia, konverzia formátu objektu alebo dolnopriepustné filtrovanie sú jednými z možností, ako aktívne kontrolovať digitálne objekty. Avšak väčšina pasívnych dozorcov deteguje stego-objekty analýzou štatistických vlastností správ. [39]

Bezpečnosť steganografie (alebo aj *stego-bezpečnosť*) je zaistená, ak je možné zaručiť, že dozorca nie je schopný – aktívnym ani pasívnym útokom – naplniť nasledujúce ciele: [26]

1. *Detegovať krycí objekt ako podozrivý* – cieľ so zameraním na odhalenie existencie tajnej komunikácie z pohľadu krycieho objektu, čo je v súčasnosti východiskovým bodom steganalýzy.

2. *Extrahovať tajnú informáciu* – cieľ so zameraním na odhalenie existencie tajnej komunikácie z pohľadu tajnej informácie. To znamená, že nejde o detekciu stego-objektu ako celku, ale o snahu z neho skrytú informáciu priamo extrahovať. Je to však možné len za podmienok úplného prístupu k potenciálnemu stego-objektu a nejakých počiatočných znalostí o použitej metóde steganografie. Takto je možné priamo určiť prítomnosť steganografie.

Existuje alternatívna definícia, ktorá vyžaduje, aby relatívna entropia[1] medzi stego-objektmi a nezávislými identicky distribuovanými vzorkami z nejakého rozdelenia pravdepodobnosti krycích objektov, bola malá. [21]

2.1.3 Útoky steganalytických systémov

Útoky steganalýzy je možné klasifikovať do štyroch úrovní podľa útočníkom nadobudnutých znalostí o steganografickom systéme. Platí, že čím viac znalostí o útočiacom cieli môže útočník získať, tým ľahšie dosiahne cieľ útoku – útok vyššej úrovne. Čím vyššej úrovni útoku dokáže steganografický systém odolať, tým vyššiu úroveň stego-bezpečnosti dosahuje. Definujeme nasledujúce úrovne útokov: [26] [15]

1. **Útok na stego-objekt** (angl. *Stego Only Attack – SOA*) – Ide o útok primárnej úrovne, pri ktorom útočník disponuje len potenciálnym stego-objektom. Avšak prístup k pôvodnému kryciemu objektu spred vloženia tajnej informácie nemá. Útočník môže použiť metódu štatistickej analýzy na modelovanie rozloženia bežne sa vyskytujúcich krycích objektov, čím následne môže detegovať prítomnosť steganografie, alebo iné bežné metódy steganalytických systémov.

2. **Útok známeho krycieho objektu** (angl. *Known Cover Attack – KCA*) – Predstavuje útok druhej úrovne. V tomto útoku má útočník okrem predchádzajúcich znalostí z prvej úrovne (SOA) aj dvojicu pôvodný krycí objekt a jemu zodpovedajúci stego-objekt. Okrem metód z prvej úrovne môže analyzovať práve túto dvojicu a pokúsiť sa prísť na použitý steganografický algoritmus.

3. **Útok vybraného stego-objektu** (angl. *Chosen Stego Attack – CSA*) – Útok tretej úrovne, kde útočník, okrem znalostí z druhej úrovne (KCA), môže praktizovať proces vkladania a extrakcie súčasného steganografického systému a sledovať tak zmeny krycieho objektu, pričom sa snaží spárovať vytvorený stego-objekt s potenciálnym stego-objektom. Ide o znalosť steganografického algoritmu, pričom je však stále steganografický kľúč tajný.

[1] https://towardsdatascience.com/information-entropy-c037a90de58f

4. **Adaptívny útok vybraného stego-objektu** (angl. *Adaptive Chosen Stego Attack – ACSA*) – Posledná štvrtá úroveň útokov na steganografický systém, predstavuje opakovaný pokus o úspech v rámci tretej úrovne (CSA).

Úroveň SOA je možné považovať za pasívny útok, pričom ostatné (vyššie) úrovne za aktívny. Všetky tieto úrovne sa sústreďujú na krycie objekty bez akejkoľvek znalosti o tajnej správe, pretože práve krycie objekty sú najčastejšie útočníkom odchytené a analyzované. [26]

2.1.4 Steganografia vs. vodotlač

Utajovanie informácií pomocou steganografie a vkladanie *digitálneho vodoznaku* (ďalej len *vodoznak*) pomocou *digitálnej vodotlače* (ďalej len *vodotlač*) je dobre zavedený a rozvíjajúci sa vedný odbor. Vodotlač je špeciálna forma steganografie, a preto s ňou veľmi úzko súvisí. Medzi jej časté aplikácie patrí: [17] [39]

- označovanie,
- ochrana autorských práv,
- ochrana integrity (neoprávnená manipulácia),
- monitorovanie a
- podmienený prístup.

Pri vodotlači sa skrytá informácia týka nosného objektu a poskytuje o ňom dodatočné informácie alebo sa týka jeho vlastností. Stego-objekt je zároveň primárnym objektom komunikačného kanála a o prítomnosti vodoznaku používatelia vedomosť majú, resp. môžu mať. V steganografii zvyčajne skrytá informácia nijako nesúvisí s nosným objektom, avšak prostredníctvom neho sa skrytá informácia odovzdáva. V tomto prípade je mimoriadne dôležité, aby skrytá správa odhalená nebola, keďže ona samotná je primárnym objektom komunikačného kanála. Pri oboch prístupoch je dôležité zachovať nepostrehnuteľnosť a robustnosť, čo výrazne ovplyvňuje vstavanú kapacitu pre vložené informácie. [39]

Kým steganografia je zraniteľná aj voči pasívnemu útoku, vodotlač môže byť ohrozená len aktívnym. Jej bezpečnosť je prelomená v momente, ak sa útočníkovi podarí vodoznak sfalšovať, zničiť alebo ním manipulovať. [26]

2.1.5 Steganografia vs. kryptografia

Najbezpečnejším spôsobom ako ochrániť informácie v súkromí je tranformácia samotných údajov do inej formy. Transformované údaje pochopia len používatelia, ktorí ich dokážu transformovať späť do pôvodnej formy. Takýto spôsob ochrany informácií sa nazýva *šifrovanie* (alebo *kryptografia*). [7] [43]

Hlavnou nevýhodou tohto prístupu je fakt, že existencia údajov nie je tajná. Dáta, ktoré boli zašifrované síce sú nečitateľné, ale stále existujú ako dáta. Široko dostupné šifrovacie algoritmy sú dnes veľmi sofistikované a ich bezpečnosť môže byť preukázaná známymi zložitými matematickými problémami. Preto je veľmi obtiažne takéto dáta dešifrovať používateľom, ktorému neboli adresované. Ak by však zručný používateľ dostal dostatok času a počítačového výkonu, mohlo by sa mu to napokon podariť. Riešením tohto problému je práve steganografia. [7] [47] [43]

Zašifrované informácie je ťažšie odlíšiť (v kontexte steganalýzy), na rozdiel od prirodzene sa vyskytujúcich digitálnych objektov (napr. obyčajný text) na nosnom médiu. Preto

bezpečnejším variantom v súkromnej/tajnej komunikácii je kombinácia kryptografie a steganografie – viacúrovňové zabezpečenie. Pri použití tajného steganografického kľúča, alebo aj pri jeho kombinácii s kryptografickým kľúčom, by sa mal uplatniť *Kerckhoffsov princíp*. Ten tvrdí, že kľúče predstavujú vstupné parametre pre steganografický algoritmus, a preto je bez ich vedomosti tajná správa v absolútnom bezpečí. To znamená, že samotný steganografický algoritmus je možné odtajniť, lebo len disponovaním dešifrovacieho kľúča je možné rozhodnúť, či načítané bity sú skutočnou ukrytou správou. [47] [43] [26]

2.1.6 Začiatky a vývin steganografie

Steganografia a utajovanie informácií nie sú novými praktikami. Prvýkrát bola praktizovaná počas starovekého Grécka, kde sa hovorí o tetovaní oholených hláv poslov (obrázok 2.4). Následne sa počkalo kým im dorástli vlasy a mohli byť poslaní osobne doručiť skrytú správu. Keď posol dorazil k príjemcovi tajnej správy, ten mu oholil hlavu a správu si mohol prečítať. [23] [39]

Obr. 2.4: **Praktika steganografie starovekého Grécka** – tetovanie oholených hláv poslov. (Prevzaté z [30] a [19])

Silu a potenciál steganografických techník ukazujú starovekí Číňania. V čase dynastie Jüan (1280–1368 n. l.) starovekej Číne vládli Mongoli. Slávny príbeh z tohto obdobia hovorí o úspešnom povstaní Číňanov Han. Pri príležitosti každoročného sviatku v polovici jesene, Číňania upiekli tzv. mesačné koláče, pričom sa v týchto krycích predmetoch ukrývala tajná správa o podrobnom pláne útoku. Plánovaná vzbura Číňanov bola vďaka informovanosti skrz koláče napokon úspešná a zvrhla mongolský režim. [39]

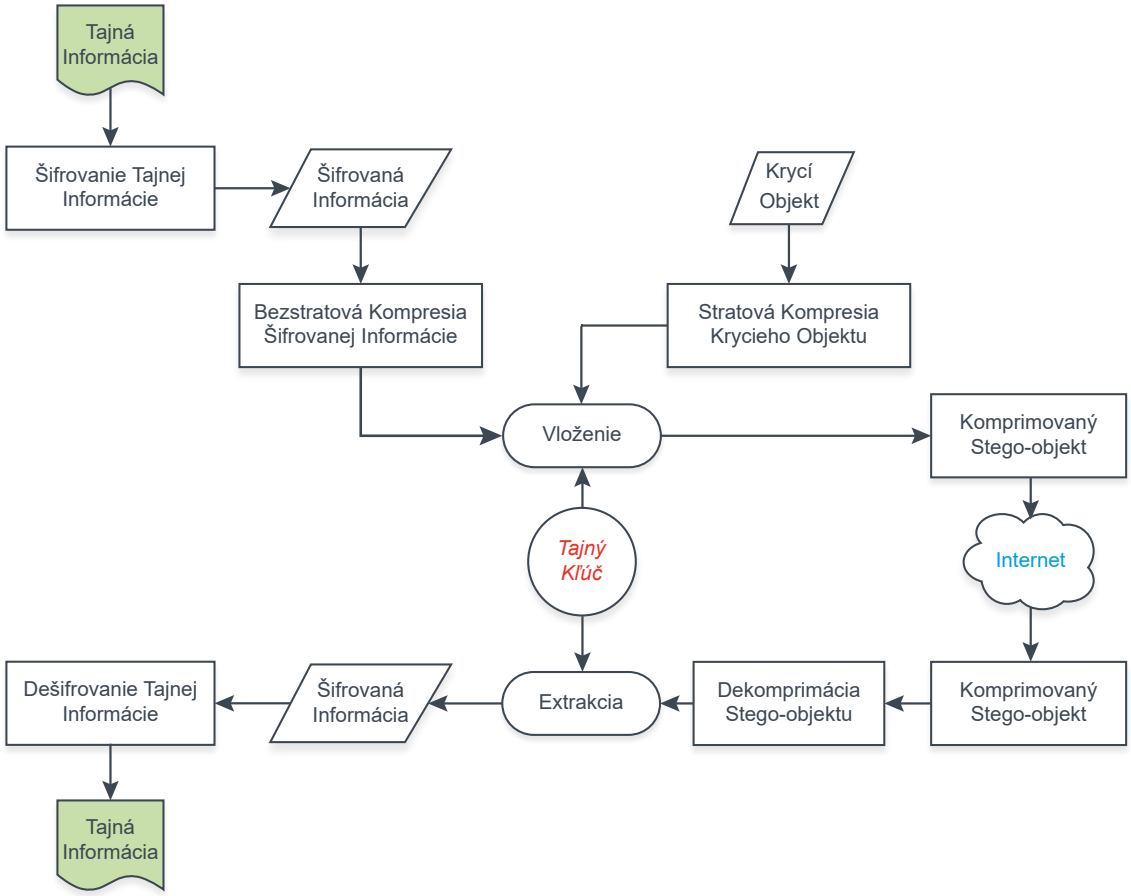
Techniky utajovania informácií sa veľmi spájajú s obdobím druhej svetovej vojny, kedy nacistické Nemecko prišlo v roku 1941 s úplne novou technikou. Mikrobodky spočívali v tom, že tajná správa o veľkosti papiera bola odfotografovaná a zmenšená na veľkosť tlačenej bodky. Týmto spôsobom bolo možné skrývať nielen textové správy ale aj obrázky či kresby. [23] [39]

2.1.7 Kompresia dát v steganografii

Úroveň zabezpečenia steganografie je možné zvýšiť použitím techniky, ktorá je dôležitou súčasťou informačnej bezpečnosti. *Kompresia dát* má za úlohu zmenšiť veľkosť digitálneho objektu, čím sa tajná správa ľahko skryje. Dáta sú po skomprimovaní bezpečnejšie a ľahšie sa s nimi manipuluje. Existujú dva typy techník kompresného algoritmu: [46]

- **Bezstratová kompresia** – hľadá dlhé reťazce kódu a vytvorí z nich alternatívne, kratšie reťazce. Nedochádza pri tom k žiadnej strate dát, a preto je možné rekonštruovať komprimované dáta do pôvodnej formy, ktorá je presne rovnaká ako predtým.
- **Stratová kompresia** – hľadá časti kódu, ktoré nie sú primárne zaujímavé pre ľudské vnímanie a odstráni ich. Keďže kompresný pomer tejto techniky je vyšší, bežne sa používa na veľké multimediálne objekty, ktoré je nutné veľkostne zmenšiť (obrázky, videosúbory, . . .).

Nasledujúci obrázok 2.5 znázorňuje proces steganografie za použitia kryptografie a kompresie.



Obr. 2.5: **Proces viacúrovňovej steganografie zahrňujúci kryptografiu a kompresiu** – Najprv sa tajná informácia zašifruje a následne skomprimuje. Súbežne s tým sa skomprimuje aj krycí objekt a aplikuje sa steganografia za použitia tajného kľúča. Tento komprimovaný súbor môže byť poslaný cez internet na miesto určenia. Prichádzajúce zakódované bity sú príjemcom dekomprimované, použitím steganografického kľúča sa tajná informácia extrahuje a na záver sa dešifruje. (Prevzaté a upravené z [46])

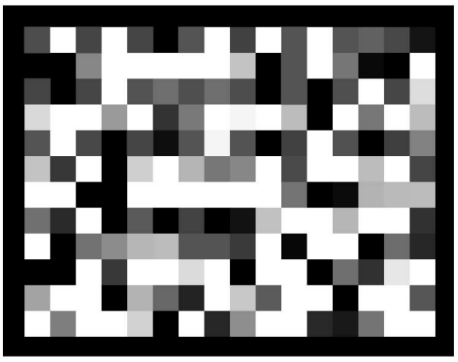
2.1.8 Použitie steganografie

Pre použitie steganografie sú najvhodnejšie digitálne objekty s vyšším stupňom *redundancie*. Redundancia je definovaná ako počet bitov, ktoré sú duplicitné alebo navyše od tých, ktoré sú požadované pre nevyhnutne presnú reprezentáciu objektu. Po odstránení redundantných bitov sa objekt nezmení. Digitálne obrázky alebo zvukové nahrávky obsahujú veľa redundantných bitov v podobe šumu, preto sú veľmi dobrými krycími objektmi. Avšak je žiadúce počítať s tým, že v tomto prípade by mohol byť stego-objekt skomprimovaný, čo by pravdepodobne tajnú správu poškodilo. Výhoda je na strane steganografa, ak dopredu vie, aká technika kompresie sa použije a či vôbec. [39] [47] [3]

2.1.9 Limity a obmedzenia digitálnej steganografie

Steganografia a šifrovanie napĺňajú osobitné ciele, avšak sú obmedzené rovnakým predpokladom. Predtým ako sa odosielateľ s príjemcom dohodne na komunikácií skrytým kanálom, musia si súkromne dohodnúť metódu steganografie, príp. steganografický kľúč. Rozdiel je v tom, že ak príjemcovi príde šifrovaná správa, okamžite vie, že bola použitá kryptografia, ale pri použití steganografie na to sám pravdepodobne nepríde. Šanca sa zmenšuje s vyšším počtom krycích objektov, pretože aj keby príjemca vedel o použitej steganografii, nevie kde konkrétne má skrytú informáciu hľadať. [7] [47]

Ďalším obmedzením je *integrita* krycieho objektu, ktorá má výrazný vplyv na jeho kapacitu. Platí, že čím menej obmedzení je pre integritu krycieho objektu, tým vyššiu kapacitu má – potenciál na skrytie údajov. Napr. integrita textu z obrázka 2.2 je obmedzená použitým jazykom a témou samotného textu. Naopak oveľa menšie obmedzenie platí pre integritu obrázka 2.6 pripomínajúceho statický obraz televízie. Podiel informácií, ktorý je možné ukryť do tohto objektu je mnohonásobne vyšší. Jediné, čo by mohlo pri tomto obrázku vzbudzovať pozornosť je jeho výpovedná hodnota. [7]



Obr. 2.6: **Statický obraz televízie** – ukážka nosného objektu steganografie s veľmi nízkym obmedzením integrity. (Prevzaté z [7])

2.2 Skrývanie dát v texte

Techniky *textovej steganografie* sú založené na použití písaného prirodzeného jazyka. Takýto digitálny textový súbor ukrýva tajné informácie, čím slúži ako krycí objekt. Ide o asi najťažší typ steganografie, pretože textový súbor obsahuje malú redundanciu dát v porovnaní s obrázkom, videom či zvukom. Navyše, štruktúra textového súboru je totožná s jej

vzhľadom, na rozdiel od iných digitálnych objektov (napr. obrázkov), preto je textová forma dát náchylnejšia na útok. Existujú tri základné kategórie metód: [18] [38] [42]

1. *Metódy založené na formáte* (angl. *Format-Based Methods*)
2. *Náhodné a štatistické metódy* (angl. *Random and Statistics Methods*)
3. *Lingvistické metódy* (angl. *Linguistic Methods*)

V prípade vyššieho záujmu je možné nahliadnuť do prílohy B, kde sú špecifikované vyššie zmienené kategórie metód spolu so základnými príkladmi ich techník. Keďže príloha B približuje len základné techniky textovej steganografie, je tiež možnosťou – pre obšírnejší prehľad techník – nahliadnuť do literatúry [29].

2.3 Skrývanie dát v obrázkoch

Najpopulárnejším druhom steganografie je *obrazová steganografia*, pretože obrázky majú veľkú bitovú redundanciu a ľahko sa šíria internetom. Navyše, takéto metódy často vkladajú informácie ako šum, ktorý je takmer nemožné ľudským zrakom spozorovať.

Vo všeobecnosti možno techniky obrazovej steganografie rozdeliť do dvoch skupín podľa spôsobu ukrytia informácie. Prvá skupina mení obraz a druhá modifikuje formát obrazového súboru. Druhá skupina techník je menej robustná. Hlavnú úlohu v tejto steganografii má kompresia, keďže obrazové súbory sú zväčša veľmi veľké. Preto je potrebné vyvíjať metódy odolné voči takémuto útoku. Metódy modifikujúce obraz sa delia na: [48] [20]

1. *Metódy priestorovej domény* (angl. *Spatial Domain Methods*)
2. *Metódy transformačnej domény* (angl. *Transform Domain Methods*)
3. *Metódy rozprestretého spektra* (angl. *Spread Spectrum Methods*)
4. *Štatistické metódy* (angl. *Statistical Methods*)
5. *Techniky skreslenia* (angl. *Distortion Techniques*)

Druhou skupinou sú techniky zahŕňajúce *vkladanie súborov* a *vkladanie paliet*. Taktiež existujú ďalšie metódy, ktoré pozmeňujú prvky v obraze, a to *techniky generovania obrazu* a *techniky úpravy obrazových prvkov*. Špeciálnym typom techník priestorovej a transformačnej domény je *adaptívna steganografia*. V tejto podkapitole sú priblížené metódy prvej skupiny modifikujúce obrazový súbor. [48] [20]

Pri vyššom záujme sa ponúka možnosť nazrieť do prílohy B. Jej obsahom je špecifikácia aj vyššie spomenutých skupín metód obrazovej steganografie so základným zhrnutím niektorých techník týchto skupín.

2.4 Skrývanie dát vo zvuku

Keď je vložená tajná informácia do digitalizovaného zvukového signálu, ide o použitie techniky *zvukovej steganografie* (alebo aj *audiosteganografie*). Takéto vloženie informácie vedie k miernej zmene binárnej sekvencie zodpovedajúceho zvukového súboru (napr. formáty MP3, WAV, ...). Keďže ľudský sluch je oveľa citlivejší ako ľudský zrak, je o to zložitejšie

vložiť informáciu do zvukového súboru ako do obrazového. Preto ju tieto metódy vkladajú ako šum s frekvenciou mimo dosahu ľudského sluchu. Avšak v ich prospech hrá fakt, že zvukové signály majú charakteristickú redundanciu a nepredvídateľný charakter, vďaka čomu sú ideálne pre tajnú komunikáciu. Všeobecne môžeme metódy kategorizovať do dvoch skupín: [8] [35]

1. *Metódy časovej domény* (angl. *Temporal Domain Methods*)
2. *Metódy transformačnej domény* (angl. *Transformation Domain Methods*)

Príloha B je k dispozícii pre prípadný záujem o špecifikáciu skupín metód zvukovej steganografie. Súčasťou prílohy sú aj jej základné techniky.

2.5 Skrývanie dát vo videu

Techniky *videosteganografie* využívajú ako krycí objekt videosúbor, ktorý má veľký kapacitný potenciál pre tajné informácie, pretože obsahuje vysoký stupeň priestorovej a časovej redundancie. Navyše, vďaka pokroku v oblasti internetu a multimediálnych technológií, sa videosúbory stali obľúbenými stego-objektmi. Takýto súbor je menej náchylný na steganalýzu, keďže video pozostáva zo série po sebe idúcich a rovnako časovo rozmiestnených statických obrázkov, ktoré môžu byť kombinované so zvukom a textom, do ktorých môžu byť takisto zakódované informácie. [32] [24]

Vo všeobecnosti ide o rozšírenie obrazovej steganografie, a preto je viacero techník obrazovej steganografie použiteľných aj na videá. Keďže je obsah videosúboru dynamický, pravdepodobnosť odhalenia skrytých informácií je nižšia ako pre obrazové objekty. Existuje však veľa efektívnych útokov na videosúbor, ako napr. stratová kompresia, zmena formátu, pridávanie či odstraňovanie snímok počas spracovania videa alebo zmena frekvencie snímok. [37]

Videosteganografiu je možné použiť v rôznych užitočných aplikáciách. Môže ísť o komunikáciu vojenských a spravodajských agentúr, korekciu chýb videa počas prenosu alebo môžu byť metódy tejto steganografie použité aj na prenos dodatočných informácií k videu (napr. titulky) bez potreby zväčšenia šírky pásma. [37]

Techniky videosteganografie sa dajú klasifikovať podľa kompresie na *komprimované* a *nekomprimované* (angl. *raw*) metódy [32]. Iná klasifikácia je založená na doméne vkladania informácie rozlišuje metódy *priestorovej domény* a *transformačnej domény*. Avšak táto podkapitola vychádza z [37], kde je uvedená nasledujúca klasifikácia:

1. *Substitučné metódy* (angl. *Substitution Methods*)
2. *Metódy transformačnej domény* (angl. *Transform Domain Methods*)
3. *Adaptívne metódy* (angl. *Adaptive Methods*)
4. *Metódy založené na formáte* (angl. *Format-Based Methods*)
5. *Metódy generujúce krycí videosúbor* (angl. *Cover Generation Methods*)

Táto klasifikácia je pre prípadných záujemcov zhrnutá a spísaná v prílohe B, kde sa tiež nachádzajú základné príklady techník videosteganografie.

2.6 Skrývanie dát v internetovej sieti

Ďalším typom digitálnej steganografie je *sieťová steganografia* (alebo aj *protokolová steganografia*), ktorej sa niekedy hovorí *steganografia 2.0* [24]. Jej techniky využívajú sieťové protokoly a dátové pakety ako krycie objekty. Ich výhodou je ťažká detegovateľnosť paketov obsahujúcich tajnú informáciu. Vo vrstvách modelu ISO/OSI[2] je možné spozorovať viacero skrytých kanálov. Možnosťou je využiť niektoré polia hlavičky TCP/IP paketu alebo iné protokoly transportnej vrstvy (napr. UDP, ICMP, ...). Mohlo by ísť napr. o skrytie informácií do tzv. bloku výplne (angl. *padding*) pre zarovnanie bitov v hlavičke paketu. Keďže žiadne zmysluplné dáta v tejto časti paketu očakávané nie sú, je vysoká pravdepodobnosť, že informácia odhalená nebude. Zaujímavou možnosťou je aj vyvolanie tzv. retransmisie, kedy sa úspešne prijatý paket úmyselne nepotvrdí. Potom tento opakovane prenášaný paket nesie tajné informácie namiesto pôvodných. [37] [7]

2.7 Iné možnosti ukrytia dát

Konvenčná steganografia sa zameriava na nepostrehnuteľnosť a nedetegovateľnosť, pretože jej hlavným cieľom je navrhnúť metódy imúnne voči steganalýze. *Nulová-steganografia* (angl. *Zero-steganography*) je vysoko nepostrehnuteľná, nedetegovateľná technika skrývania informácií, pretože počas celého procesu nijako nemodifikuje krycí objekt, a preto je steganalýza absolútne bezpredmetná. Nulová-steganografia je zabezpečená tajným kľúčom, ktorý je vytvorený na základe určitého vzťahu medzi krycím objektom, maticou chaosu a samotnou tajnou informáciou. Extrakcia tajnej informácie je založená na vzťahu medzi krycím objektom, tajným kľúčom a maticou chaosu. Okrem nepostrehnuteľnosti a nedetegovateľnosti ponúka táto technika bezpečnosť (zvýšenú najmä vďaka použitiu mapy chaosu [4]) a dostatočnú kapacitu. [9]

Dôkazom, že vedecký výskum v oblasti steganografie je v plnom prúde je aj nasledujúca myšlienka tzv. *Steganografie s nesprávnym nasmerovaním* (angl. *Misdirection steganography*). Ide o techniku, pri ktorej sú tajné informácie chránené aj v prípade, že útočník má podozrenie o ich vložení do krycieho objektu. Používajú sa dva druhy vkladajúcich tajných informácií do jedného stego-objektu, a to: [31]

- **Skutočné** – sú to informácie, ktoré sú určené na prenos v tajnosti a sú bezpečne skryté v krycom objekte.
- **Falošné/klamlivé** – ide o informácie, ktoré môžu byť útočníkovi známe, pričom tým chránia skutočné informácie.

Avšak vzťah medzi týmito dvomi druhmi informácií žiaden nie je. Cieľom techniky je preniesť skutočné informácie tak, že útočníkovi umožní, aby venoval pozornosť falošným informáciám – tie ho nasmerujú nesprávne. Proces vkladania skutočných tajných informácií závisí od toho, ako vložiť tie falošné. Inštrukcie potrebné k extrakcii falošných tajných informácií (alebo samotné falošné informácie) môžu, ale nemusia, byť potrebné pre extrakciu tých skutočných. Taktiež je možné skryť skutočné tajné informácie v rámci falošných. [31]

[2] https://www.techtarget.com/searchnetworking/definition/OSI

Kapitola 3

Digitálna steganografia spustiteľných súborov a ich analýza

Jednou z možností, kam ukryť tajné dáta môže byť aj program. Táto kapitola začína analýzou formátov spustiteľných súborov ELF (podkapitola 3.1) a PE (podkapitola 3.2). Kapitola pokračuje definíciou spôsobov, ako je možné ukryť tajné dáta v spustiteľných súboroch (podkapitola 3.3). Na záver kapitoly (podkapitola 3.4) sú zhrnuté existujúce programové vybavenia z tejto oblasti.

3.1 Analýza formátu ELF

ELF[1] definuje štruktúru pre binárne súbory, knižnice a súbory jadra operačného systému (ďalej len OS). Ide o *objektové súbory*, ktoré sú vďaka svojej binárnej reprezentácii spúšťané priamo na procesore. ELF bol pôvodne vyvinutý a publikovaný spoločnosťou UNIX System Laboratories ako súčasť aplikačného binárneho rozhrania (ďalej len ABI[2]). Výbor pre štandardy rozhrania nástrojov TIS[3] zvolil vtedy vyvíjajúci sa štandard ELF ako formát prenosného objektového súboru, ktorý fungoval v prostredí IA-32[4] pre rôzne OS. [45]

Štandard ELF poskytuje vývojárom súbor definícií ABI, ktoré sú prítomné vo viacerých OS. Potrebný počet rôznych implementácií rozhrania sa znižuje, čím sa zníži aj potreba prekódovať a preložiť kód. Takýmto zefektívnením vývoja software sa ELF stáva výkonnejší a flexibilnejší binárnym formátom oproti starším a.out a COFF[5]. Existujú tri hlavné typy objektových súborov formátu ELF: [10] [45]

- **Spustiteľný súbor** (angl. *Executable File*) – Spustiteľný súbor obsahuje kód a dáta vhodné na spustenie, tiež špecifikuje rozloženie pamäte procesu.
- **Premiestniteľný súbor** (angl. *Relocatable File*) – obsahuje kód a dáta, ktoré sú vhodné na prepojenie s inými objektovými súbormi na vytvorenie spustiteľného súboru alebo súboru zdieľaného objektu.
- **Súbor zdieľaného objektu** (angl. *Shared Object File*) – známy tiež ako zdieľaná knižnica, obsahuje kód a dáta vhodné na prepojenie v dvoch kontextoch. V prvom

[1] Spustiteľný a prepojiteľný formát (angl. *Executable and Linkable Format – ELF*)
[2] Aplikačné binárne rozhranie (angl. *Application Binary Interface – ABI*)
[3] Štandardy rozhrania nástrojov (angl. *Tool Interface Standards – TIS*)
[4] 32-bitová architektúra Intel (angl. *32-bit Intel Architecture – IA-32*)
[5] Formát súboru bežného objektu (angl. *Common Object File Format – COFF*)

ho môže tzv. *linker* spracovať s inými premiestniteľnými súbormi a súbormi zdieľaných objektov na vytvorenie nového objektového súboru. V druhom kontexte ho tzv. *dynamický linker* kombinuje so spustiteľným súborom a inými zdieľanými objektmi, aby vytvoril obraz procesu – reprezentácia spustiteľného súboru v pamäti po tom, čo je do nej načítaná [34].

Formálna špecifikácia zabezpečuje správnosť interpretácie základných strojových inštrukcií OS. Súbory ELF sú zvyčajne výstupom prekladača alebo linkera, no používajú sa aj pre samotné jadro a moduly jadra OS Linux. [10]

3.1.1 Štruktúra súborov ELF

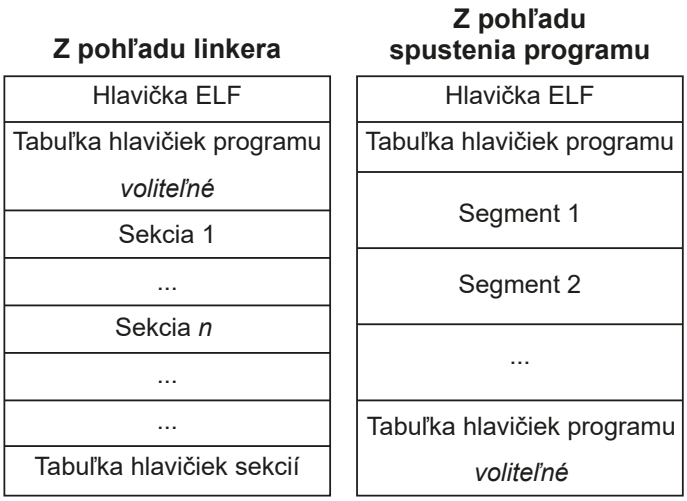
Keďže je formát ELF rozšíriteľný a používa sa pre viacero typov binárnych súborov, ich štruktúra sa mierne líši. Všeobecne pozostáva ELF z dvoch hlavných častí:

1. *Hlavička ELF* (angl. *ELF Header*)
2. *Dáta súboru* (angl. *File Data*)

Dáta súboru je ešte možné rozdeliť na tri menšie časti:

1. *Tabuľka hlavičiek sekcií a sekcie* (angl. *Section Headers Table and Sections*)
2. *Tabuľka hlavičiek programu a segmenty* (angl. *Program Headers Table and Segments*)
3. *Užitočné dáta* (angl. *Payload*)

Existujú dva komplementárne pohľady na súbor ELF (znázorňuje obrázok 3.1). Jeden je použitý pri tvorbe programu linkerom a druhý pri spustení programu, od čoho závisí aj použitie vyššie definovaných hlavičiek. [10] [45]



Obr. 3.1: **Komplementárne pohľady na súbor ELF** – Hoci obrázok ukazuje tabuľku hlavičiek programu hneď za hlavičkou ELF a tabuľku hlavičiek sekcií za sekciami, v skutočných súboroch sa to môže líšiť. Okrem toho, sekcie a segmenty nemajú definované poradie. Pevnú pozíciu v súbore má len hlavička ELF. (Prevzaté a preložené z [45])

Hlavička ELF

V *hlavičke ELF* sa nachádzajú informácie o súbore. Je povinná, pretože zabezpečuje správnu interpretáciu dát počas prepájania (angl. *linking*) a spustenia. Začína sa vždy rovnakými štyrmi bajtmi nazývanými *magická konštanta*. Táto konštanta definuje hexadecimálne formát súboru, pričom sa začína ustáleným prefixom. Magická konštanta vyzerá následovne: `7f` `45` = **E** `4c` = **L** `46` = **F**. Za magickou konštantou sa nachádzajú ďalšie bajty, pričom význam niektorých z nich je vysvetlený vzápätí: [10] [45]

- **Trieda** (angl. *Class*) – bajt nachádzajúci sa hneď za magickou konštantou. Trieda definuje architektúru súboru (`0x01` pre 32-bitovú a `0x02` pre 64-bitovú). Súbor využívajúci 64-bitovú architektúru je na výstupe príkazu `readelf` v prostredí Linux označený ako ELF64.
- **Dáta** (angl. *Data*) – Keďže rôzne procesory zaobchádzajú s dátovými štruktúrami a strojovými inštrukciami rôzne, je dôležité informovať procesor o tom, ako interpretovať zostávajúce objekty v súbore. Pre tento účel slúži bajt definujúci dátové pole. Hodnota `0x01` predstavuje LSB[6], známy aj ako *Little-Endian*. Hodnota bajtu `0x02`, naopak, predstavuje MSB[7] známy aj ako *Big-Endian*.
- **Verzia** (angl. *Version*) – číslo verzie ELF formátu. Doposiaľ existuje len jedna verzia, a tak je tento bajt vždy nastavený na hodnote `0x01`.
- **OS/ABI** – Súčasné OS majú veľkú časť ABI spoločnú. Existujú však prípady, kedy sa môžu niektoré OS v ich funkciách líšiť. Tento bajt špecifikuje, aké ABI sa používa, čím dáva najavo, aké funkcie môže OS a aplikácie od súboru ELF očakávať. Ak sa nepoužíva žiadne špeciálne rozšírenie ABI, hodnota bajtu je `0x00` s označením UNIX – System V.
- **Verzia ABI** (angl. *ABI version*) – V prípade potreby, bajt špecifikuje číslo verzie ABI. Ak sa nepoužíva žiadne rozšírenie ABI, hodnota je nastavená na `0x00`.
- **Typ** (angl. *Type*) – Dva bajty signalizujúce, o aký typ súboru sa jedná. Môže ísť o spustiteľný súbor (EXEC) – `0x0002`, súbor jadra (CORE) – `0x0004` atď.
- **Stroj** (angl. *Machine*) – Dva bajty odhadujúce typ stroja, ktoré určujú architektúru cieľovej inštrukčnej sady (napr. IA-64 – `0x0032`, AMD64 – `0x003e`, ...).

Skutočná veľkosť niektorých štruktúr objektových súborov je v hlavičke ELF definovaná, preto je možné ich zväčšovať či zmenšovať. Ak sa zmení formát objektového súboru, program môže naraziť na štruktúry, ktoré sú väčšie alebo menšie, ako sa očakávalo. Programy tak môžu ignorovať niektoré informácie. Spracovanie „chýbajúcich“ informácií závisí od kontextu a môže byť špecifikované definovaním rozšírení. [45]

Tabuľka hlavičiek sekcií a sekcie

Tabuľka hlavičiek sekcií (ďalej len *THS*) disponuje informáciami, ktoré popisujú jednotlivé *sekcie* súboru. V THS je zaznamenaná každá z nich, pričom položky THS poskytujú informácie, ako názov sekcie, veľkosť atď. Na špecifických pozíciách sa v THS nachádzajú aj

[6] Najmenej významný bajt (angl. *Least Significant Byte – LSB*)
[7] Najviac významný bajt (angl. *Most Significant Byte – MSB*)

záznamy, ktoré sú rezervované a objektový súbor pre nich nemá žiadne sekcie. Významnými dátami o THS – ktoré sú uložené v hlavičke ELF – sú: [45]

- `e_shoff` – udáva posun (v bajtoch) THS od začiatku objektového súboru.
- `e_shnum` – udáva počet záznamov (sekcií) THS.
- `e_shentsize` – udáva veľkosť (v bajtoch) každého záznamu THS.

Sekcie obsahujú väčšinu informácií o objektovom súbore užitočné z pohľadu prepájania objektových súborov (inštrukcie, tabuľku reťazcov, tabuľku symbolov, . . .). Z tohto dôvodu je táto časť súborov ELF povinná, len ak ide o súbory používané počas prepájania, no pre spustiteľné a iné objektové súbory je len voliteľná. Tiež platí, že každá sekcia má svoju hlavičku, ktorá ju popisuje. Môžu existovať aj hlavičky sekcií, ktoré nemajú žiadnu sekciu. Zároveň, každá sekcia je v pamäti uložená v rámci súvislého bloku bajtov (možno prázdneho). Sekcie sa nesmú prekrývať, teda žiaden bajt sa nenachádza vo viacerých sekciách zároveň. Objektový súbor môže obsahovať aj neaktívne miesta, pretože nie každý bajt súboru musí byť nutne pokrytý THS alebo sekciou. Obsah týchto častí nie je špecifikovaný. [45]

Rôzne sekcie držia riadiace a programové informácie. Súčasťou súboru môžu byť aj tzv. *špeciálne sekcie*, ktoré používa systém a ktoré majú svoje typy a atribúty. Pre ich vysoký počet sa nasledujúci zoznam obmedzuje len na tie najzaujímavejšie: [45] [10]

- **.text** – sekcia obsahuje „text“ v podobe spustiteľných inštrukcií programu. Obsah bude zabalený do segmentu s prístupovými právami na čítanie a vykonávanie. Načíta sa iba raz, pretože obsah sa nezmení.
- **.data** a **.data1** – sekcia obsahuje inicializované dáta s prístupovými právami k čítaniu aj zápisu. Dáta prispievajú k obrazu pamäte programu.
- **.rodata** a **.rodata1** – sekcia obsahuje inicializované dáta, ale s prístupovými právami len k čítaniu. Dáta zvyčajne prispievajú k nezapisovateľnému segmentu v obraze procesu.
- **.bss** – sekcia obsahuje neinicializované dáta (čítanie aj zápis), ktoré prispievajú k obrazu pamäte programu. Podľa definície systém inicializuje dáta s nulami pri spustení programu. Sekcia nezaberá žiaden súborový priestor.
- **.init** – sekcia obsahuje spustiteľný kód, ktorý prispieva k inicializačnému kódu procesu – keď program začne bežať, systém zariadi spustenie tohto kódu pred volaním hlavného vstupného bodu programu (tzv. „main“).
- **.fini** – sekcia obsahuje spustiteľný kód, ktorý prispieva ku kódu ukončenia procesu – keď sa program správne ukončí, systém zariadi spustenie tohto kódu.

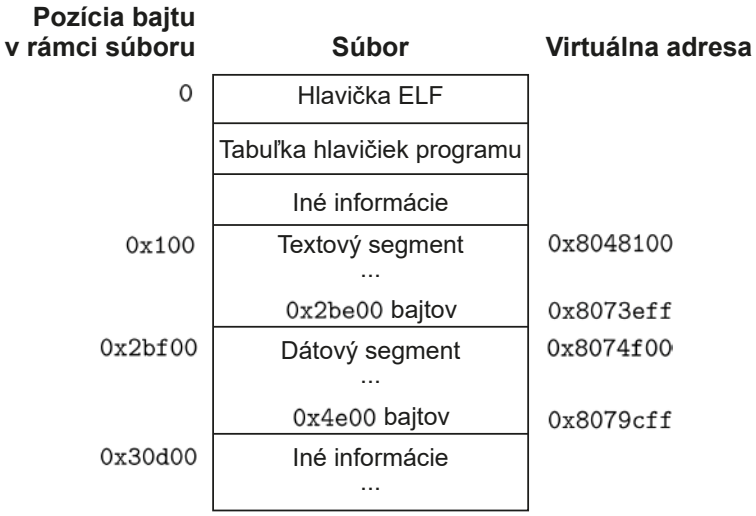
Názvy sekcií s prefixom (bodkou) sú vyhradené pre systém, no aplikácie ich môžu používať tiež, ak to pre ne má nejaký význam. Avšak odporúčaním je (pre aplikácie) používať názvy sekcií bez tohto prefixu, aby nedošlo ku konfliktu mien. Objektové súbory formátu ELF umožňujú definíciu vlastných sekcií, pričom názvy sekcií byť unikátne nemusia. Existuje konvencia pre názvy sekcií, ktoré sú určené pre konkrétnu architektúru procesora. V tomto prípade je názov vo vzore *.ARCH.psect*, kde „ARCH“ je používaný názov architektúry v hlavičke ELF (`e_machine`) a „psect“ je názov sekcie. Potom ide o sekciu „psect“ definovanú architektúrou „ARCH“. [45]

Tabuľka hlavičiek programu a segmenty

Vo všeobecnosti ide o voliteľnú časť súborov ELF, avšak ak ide o spustiteľné súbory alebo súbory zdieľaného objektu, tie ich obsahovať musia. *Tabuľka hlavičiek programu* (ďalej len *THP*) je primárna dátová štruktúra, ktorá lokalizuje *segmenty* v súbore a obsahuje informácie, ktoré sú potrebné na vytvorenie pamäťového obrazu programu. Jednotlivé segmenty môžu niesť informáciu o zásobníku (segment `GNU_STACK`) či obsluhe výnimiek (segment `GNU_EH_FRAME`). [45] [10]

Každý záznam THP popisuje segment alebo iné informácie, ktoré systém potrebuje na prípravu programu na spustenie. Špecifikácia veľkosti každého záznamu v THP je, podobne ako v časti 3.1.1, uvedená v hlavičke ELF v `e_phentsize` a ich počet (nula alebo viac) je uvedený v `e_phnum`. V hlavičke ELF sa tiež obdobne nachádza miesto začiatku THS (`e_phoff`) v rámci súboru. [45]

Segment objektového súboru pozostáva zo žiadnej alebo z niekoľkých sekcií. O tejto skutočnosti však THP nenesie žiadne informácie. Počet sekcií v rámci segmentu je tiež nepodstatný pre načítanie programu. Avšak prítomné musia byť rôzne informácie na vykonávanie programu, dynamické prepojenie atď. Poradie a zaradenie sekcií v segmente nie je presne definované. Textové segmenty obsahujú inštrukcie a dáta len na čítanie. Na druhú stranu, dátové segmenty obsahujú inštrukcie a dáta s možnosťou zápisu. Spustiteľný súbor rozdelený na segmenty znázorňuje obrázok 3.2. [45]



Obr. 3.2: **Príklad spustiteľného súboru ELF architektúry System V** – znázornenie štruktúry spustiteľného súboru ELF na architektúre System V vrátane virtuálnych adries a segmentov. (Prevzaté a preložené z [45])

3.2 Analýza formátu PE

Formát *PE*[8] definuje súbor špecifikácií, ktorými sa riadia všetky spustiteľné súbory PE, dynamicky pripojené knižnice (DLL[9]) a iné. Ide o štandardný formát týchto súborov rodiny OS Windows. Vychádza zo špecifikácie COFF a bol prijatý spoločnosťou Microsoft

[8] Prenosný spustiteľný súbor (angl. *Portable Executable – PE*)
[9] Dynamicky pripojená knižnica (angl. *Dynamic-Link Library*)

od vydania OS Windows NT 3.1. Odvtedy však prešiel sériou zmien, vďaka ktorým bola pridaná podpora nových funkcií. Základný dizajn formátu PE zostal nezmenený. V súčasnosti existujú dva formáty súborov PE. Pre systémy x86 je to PE32 a pre systémy x64 je to PE32+. [34] [28]

3.2.1 Štruktúra súborov PE

Všeobecne sa súbory PE skladajú z *hlavičiek* a *sekcií*. V súbore sa nachádza hneď niekoľko hlavičiek, ktoré je možné dekomponovať na menšie zmysluplné časti: [12]

1. *Hlavička MS-DOS* (angl. *MS-DOS Header*)
2. *Útržok MS-DOS* (angl. *MS-DOS Stub*)
3. *Hlavička PE* (angl. *PE Header*)
4. *Tabuľka sekcií* (angl. *Sections Table*)

Hlavička PE sa delí na tri samostatné časti, a to: [12]

1. *Podpis PE* (angl. *PE Signature*)
2. *Hlavička COFF* (angl. *COFF Header*)
3. *Voliteľná hlavička* (angl. *Optional Header*)

Ďalej popisované dátové štruktúry možno pozorovať v hlavičkovom súbore „WINNT.h“, ktorý je súčasťou Windows SDK[10].

Hlavička MS-DOS

Hlavička MS-DOS (dátová štruktúra `Image_MS-DOS_Header`) je povinnou časťou každého spustiteľného súboru PE. Najdôležitejšie sú dve položky:

- `e_magic` – Magická konštanta zaberajúca prvé dva bajty hlavičky, ktorá je vždy nastavená na hodnotu `0x4d` = **M** `0x5a` = **Z**. *MZ* je jedinečný identifikátor a predstavuje Marka Zbikowského, tvorcu MS-DOS. V spustiteľných súboroch sa nachádza od OS MS-DOS a dodnes sa používa kvôli spätnej kompatibilite s MS-DOS.
- `e_lfnew` – Posledná položka hlavičky MS-DOS (od bajtu `0x3c` v rámci súboru) nesie adresu začiatku hlavičky PE. V skutočnosti patrí táto adresa podpisu súboru PE. Je to nutné z dôvodu, že medzi hlavičkou MS-DOS a PE sa ešte nachádza tzv. *útržok MS-DOS*. Adresa je však zadaná nepriamo, a to ako poradie bajtu od začiatku súboru PE.

Všetky ostatné polia nie sú pri analýze súborov PE také užitočné, pretože len pomáhajú pri spúšťaní programov. [28] [12]

[10] Balíček na vývoj software (angl. *Software Development Kit – SDK*)

Útržok MS-DOS

Ako už bolo vyššie načrtnuté, nasleduje útržok MS-DOS. Ide o skutočný program, ktorý spúšťa systém MS-DOS pri načítaní spustiteľného súboru. Pre neskoršie OS Windows sa tu nachádza útržok programu MS-DOS, ktorý beží namiesto skutočnej aplikácie. Zvyčajne táto sekcia len vypisuje buď „Tento program sa nedá spustiť v režime MS-DOS.“, alebo „Tento program musí byť spustený pod win32.“. Táto sekcia je plne zodpovedná za správanie programu, ak by bol spustený v systéme MS-DOS a vytvorený pre spätnú kompatibilitu. Útržok MS-DOS sa nachádza bezprostredne za 64 bajtovou hlavičkou MS-DOS. Je možné tiež vytvoriť vlastný obsah tejto časti súboru. [12] [28]

Podpis PE

Spustiteľným súborom systému Windows a OS/2 bol pridaný tzv. *podpis PE*, ktorý určuje zmýšľaný cieľový OS (OS/2 alebo MS-DOS, alebo Windows NT). Ak ide o formát súboru PE v systéme Windows NT, podpis PE sa nachádza bezprostredne pred štruktúrou hlavičky PE. Naopak, vo verziách Windows a OS/2 je podpis prvou položkou hlavičky PE. V systéme Windows NT zaberá podpis štyri bajty a má tvar: `0x50` = **P** `0x45` = **E** `0x00` = **\0** `0x00` = **\0**. [28]

Hlavička COFF

Hlavička COFF (dátová štruktúra `IMAGE_FILE_HEADER`) disponuje len informáciami vyššej úrovne, ktoré hovoria, ako so súborom PE zaobchádzať. Medzi informáciami sú aj dáta, či ide o dynamickú analýzu alebo spustiteľný súbor a či jeho obraz v pamäti podporuje náhodnú základnú adresu (angl. *Randomized Base Address*). `IMAGE_FILE_HEADER` sa skladá z týchto položiek: [28] [34]

- `Machine` – Dvojbajtový identifikátor reprezentujúci architektúru procesora (napr. `0x8664` pre AMD64, `0x14c` pre IA-32, ...).
- `NumberOfSections` – Definuje počet hlavičiek sekcií a tiel sekcií v súbore PE, pričom existuje obmedzenie na 96 sekcií. Pomocou tohto počtu je možné zistiť celkovú veľkosť tabuľky sekcií, keďže každá hlavička sekcie a telo sekcie sú v súbore usporiadané postupne.
- `TimeDateStamp` – predstavuje dátum vytvorenia súboru PE.
- `PointerToSymbolTable` – obsahuje posun v bajtoch k tabuľke symbolov. Používa sa zriedka (zvyčajne vyplnené nulami), keďže táto informácia je zastaralá.
- `SizeOfOptionalHeader` – definuje veľkosť voliteľnej hlavičky.
- `Characteristics` – príznak (angl. Flag), ktorý predstavuje niektoré charakteristiky súboru pomocou preddefinovaných konštánt (napr. či ide o spustiteľný súbor, systémový súbor, informácia o výskyte ladiacich informácií v súbore, ...).

Voliteľná hlavička

Napriek názvu, *voliteľná hlavička* (dátová štruktúra `IMAGE_OPTIONAL_HEADER`) je povinnou súčasťou súborov PE pre ich spustenie. Začína hneď za hlavičkou COFF a poskytuje

podrobnejšie informácie o spustiteľnom obraze súboru PE. Má dva hlavné typy podľa architektúry systému (PE32 alebo PE32+). Tento typ je uložený magickou konštantou v prvých 22 bajtoch (`0x10b` pre PE32 a `0x20b` pre PE32+). Výhodou dizajnu tejto hlavičky je, že jej veľkosť nie je pevná. Je určená v hlavičke COFF, čo uľahčuje jej rozšírenie v budúcich úpravách formátu PE. Položky hlavičky, ktoré stoja za zmienku: [12] [34]

- `MajorLinkerVersion` – Predstavuje číslo hlavnej verzie linkera.
- `MinorLinkerVersion` – Predstavuje číslo vedľajšej verzie linkera.
- `SizeOfCode` – Veľkosť kódovej časti v bajtoch. Ak sa kód rozprestiera medzi viacero sekcií potom položka definuje súčet všetkých sekcií kódu.
- `SizeOfInitializedData` – Veľkosť inicializovanej dátovej sekcie v bajtoch alebo súčet všetkých takýchto sekcií, ak existuje viacero inicializovaných dátových sekcií.
- `SizeOfUninitializedData` – Veľkosť neinicializovanej dátovej sekcie v bajtoch. V prípade viacerých neinicializovaných dátových sekcií, súčet všetkých týchto sekcií.
- `ImageBase` – Preferovaná základná adresa v adresnom priestore procesu, na ktorý sa má priradiť spustiteľný obraz. Predvolená hodnota linkera je `0x00400000`, no je možné ju zmeniť.
- `SizeOfImage` – Definuje veľkosť adresného priestoru, ktorý sa má rezervovať pre načítaný spustiteľný obraz.

.

Veľmi dôležitá je tiež posledná položka voliteľnej hlavičky, a to *tabuľka dátových adresárov*. Počet dátových adresárov nie je stanovený a určuje ho samotná tabuľka. Príkladmi dátových adresárov sú: [34]

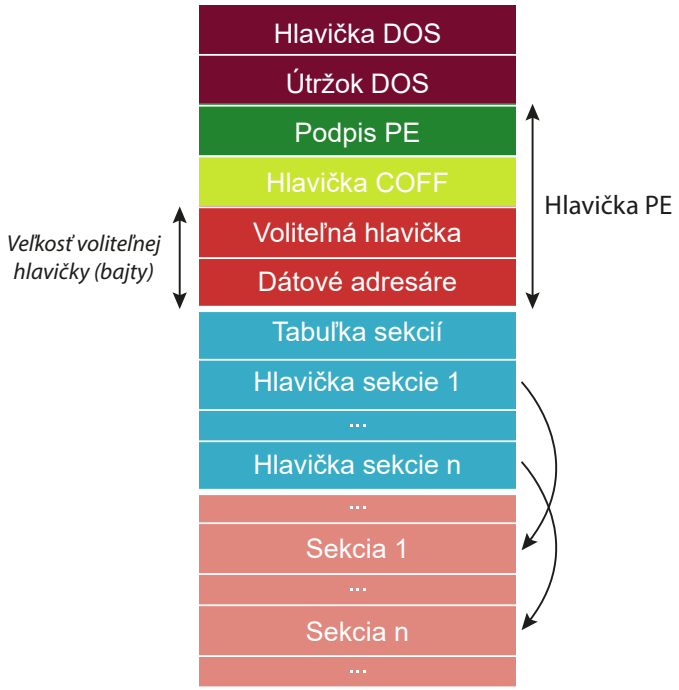
- *Importná tabuľka* (angl. *Import Table*) – Deklaruje dynamické knižnice (DLL) a funkcie, ktoré je potrebné načítať spolu so spustiteľným súborom.
- *Exportná tabuľka* (angl. *Export Table*) – Obsahuje funkcie, ktoré spustiteľný súbor sprístupňuje iným programom na použitie.
- *Tabuľku certifikátov* (angl. *Certificate Table*) – Obsahuje digitálne podpisy vývojára spustiteľného súboru.

Tabuľka sekcií a sekcie

Nasledujú *hlavičky sekcií*, ktoré sú uložené v tabuľke. Táto tabuľka sa musí nachádzať bezprostredne za voliteľnou hlavičkou (ak existuje), pretože informácia o jej umiestení nie je nikde uložená. Jej pozícia je daná výpočtom prvého bajtu po hlavičkách. Každý záznam v tabuľke – o veľkosti 40 bajtov – definuje *sekciu*, ktorá tvorí súvislú pamäťovú oblasť obrazu. Tá je buď neinicializovaná, alebo vyplnená časťami spustiteľného súboru. Sekcie obsahujú obsah súboru vrátane kódu, údajov, zdrojov a iných spustiteľných informácií. Každá sekcia má hlavičku a telo. Hlavičky sekcií majú presne definovanú štruktúru, no telám sekcií pevná štruktúra chýba. Sekcie môžu byť usporiadané takmer akokoľvek, pokiaľ hlavička obsahuje dostatok informácií o príslušnom tele sekcie. Súbor PE pre OS Windows NT má zvyčajne tieto preddefinované sekcie: [28] [34]

- **.text** – Obsahuje kód programu.
- **.bss** – Obsahuje neinicializované dáta vrátane všetkých premenných deklarovaných ako statické v rámci funkcie alebo zdrojového modulu.
- **.rdata** – Obsahuje dáta len na čítanie (reťazce, konštanty a informácie o adresári ladenia).
- **.data** – Obsahuje globálne premenné aplikácie či modulu.
- **.rsrc** – Obsahuje zdrojové informácie pre modul. Dáta sekcie sú štruktúrované do stromu zdrojov.
- **.edata** – Obsahuje informácie o funkciách exportovaných z knižnice DLL. Ak je sekcia k dispozícii, obsahuje exportný adresár na získanie informácií o exporte.
- **.idata** – Obsahuje informácie o funkciách importovaných spustiteľným súborom alebo knižnicou DLL vrátane importného adresára a tabuľky názvov adries importu.
- **.debug** – Obsahuje informácie o ladení, ktoré sú spočiatku umiestnené v tejto sekcii. Ako prostriedok na zhromažďovanie informácií o ladení na jednom mieste môžu byť pri formáte PE aj samostatné súbory (s príponou .DBG). Aj keď táto sekcia obsahuje informácie o ladení, adresáre ladenia sa nachádzajú v časti .rdata uvedenej vyššie. Každý z týchto adresárov odkazuje na informácie o ladení v tejto sekcii.

Obrázok 3.3 znázorňuje štruktúru spustiteľného súboru PE tvoriaceho vyššie rozobrané komponenty.



Obr. 3.3: **Štruktúra spustiteľného súboru PE**

3.3 Spôsoby ukrytia dát v spustiteľných súboroch

S výnimkou zdrojov použitých v tejto podkapitole je verejne dohľadateľné len malé množstvo informácií k metódam steganografie spustiteľných súborov. Veľká pozornosť sa však dlhú dobu venuje témam súvisiacim s vkladaním vodoznaku do programov. Z vlastností samotnej vodotlače a skutočnosti, že vodotlač sa snaží o vloženie dát, ktoré musia byť neodstraniteľné, no nemusia byť nedetegovateľné vyplýva, že tieto metódy nie sú použiteľné v kontexte steganografie spustiteľných súborov. [2]

Napriek relatívnemu nedostatku literatúry pre steganografiu spustiteľných súborov je možné jej techniky klasifikovať do dvoch tried, podľa princípu ich fungovania, a to na techniky *pridávajúce kód alebo časti dát do kódu* a techniky *modifikujúce vlastnosti existujúcich spustiteľných programov*. Súčasťou prvej triedy sú metódy ako: [16] [2]

- **Analýza mŕtveho kódu** – Za mŕtvy kód môže byť označená každá inštrukcia, ktorá nie je navštívená žiadnym zo všetkých možných tokov dát. Na miesta mŕtveho kódu by bolo možné vložiť dáta kódované vo forme pripomínajúcej inštrukcie strojového kódu. V tomto prípade je bezpečnosť ukrytia dát priamoúmerná bezpečnosti použitej formy pripomínajúcej inštrukcie.
- **Staticky prepojené binárne súbory** – typicky celé knižnice pripojené k programu, avšak veľká časť z funkcií týchto knižníc je vo väčšine prípadov nepoužívaná. Kódovať požadované dáta na tieto miesta by preto bolo možné podobne ako je uvedené vyššie pre mŕtvy kód.

Všetky metódy nasledujúcich sekcií boli vyvinuté pre steganografické vloženie dát do binárnych súborov programov kompatibilných s architektúrou IA-32. Metódy využívajú redundancie konkrétnej inštrukčnej sady procesorov, čo spôsobuje významnú závislosť metód na tejto sade. Keďže pracujú na úrovni strojového kódu, ide o inherentne zníženú redundanciu pre ukrytie dát v porovnaní s inými steganografiami. Uvedené metódy pracujú nad výsledným preloženým súborom, pričom znalosť zdrojového kódu nie je potrebná. [2] [16] [1]

3.3.1 Substitúcia inštrukcií

Substitúcia inštrukcií (angl. *Instruction Substitution* alebo aj *Instruction Selection*, ďalej len IS) predstavuje najznámejšiu a jednu z prvých predstavených metód svojho druhu. Takýto systém využíva sadu tried funkčne ekvivalentných inštrukcií ako redundantné bity. Dátová rýchlosť (angl. *Data Rate*) metódy IS je preto závislá od prítomnosti inštrukcií ekvivalenčných tried v rámci krycieho programu. [16]

Vo všeobecnosti môžu byť súčasťou jednej ekvivalenčnej triedy rôzne dlhé inštrukcie, avšak v tom prípade je nutné veľmi výrazným zásahom zmeniť celý spustiteľný súbor (posun všetkých následných adries skokov a volaní funkcií či zmena referencií umiestnenia rôznych dát súboru), čo celú záležitosť vkladania mimoriadne komplikuje. Preto sa pre jednoduchosť implementácia z [16] obmedzuje na ekvivalenčné triedy inštrukcií o rovnakej bajtovej dĺžky. Toto zjednodušenie prináša výhodu nezmenenej veľkosti spustiteľného súboru. Taktiež je v záujme nájsť čo najväčšie ekvivalenčné triedy, keďže jej veľkosť určuje jej bitovú šírku. Pri použití sady ekvivalenčných inštrukcií o veľkosti n je možné zakódovať $\log_2(n)$ bitov. [16]

Pri technike IS je nutné kontrolovať ekvivalenciu inštrukcií v rámci triedy v každom kontexte možnej zámeny a zabezpečiť, že nenastane situácia, kedy by takáto zámena inštrukcie mohla spôsobiť odlišné chovanie – zmeniť tok programu. Typickým rozdielom rôznych inštrukcií je odlišne nastavenie príznakov (angl. *flags*) v rámci príznakového registra.

V prípade, že jedna inštrukcia v rámci triedy modifikuje nejaký príznak odlišne ako tá zamieňaná, je nutné previesť detekciu živosti daného príznaku. Príznak je považovaný za živý, ak je jeho hodnota použitá niektorou z následujúcich inštrukcií skôr, ako dôjde k jej ďalšej modifikácii inou inštrukciou. Ako sa uvádza v [16], takéto inštrukcie sú zriedkavé, a preto strata v dôsledku živosti príznaku je malá. [16]

Analýza existujúcich implementácií metódy IS

V [16] boli analyzované binárne súbory niekoľkých distribúcií OS. Bolo zistené, že dostupná kapacita sád ekvivalenčných tried, ktoré boli použité dosahuje hodnotu $\frac{1}{110}$, čo znamená, že je možné približne zakódovať 1 bit za každých 110 bitov programového kódu. V tomto prípade sa jedná o čisto kódovú časť spustiteľných súborov, ktorá po experimentálnom meraní predstavuje 75 % celkovej veľkosti týchto súborov. Hodnota nameranej kapacity sa v rámci rôznych skúmaných OS významne nelíšila, pretože najviac používaným prekladačom je GCC naprieč väčšiny skúmaných OS. [16]

Pre porovnanie ide o vyše šesťnásobne menšiu kapacitu narozdiel od štandardnej techniky obrazovej steganografie, ktoré používajú súbory JPEG ako krycie. Jej nameraná kapacita bola v dôsledku veľkého množstva redundancie $\frac{1}{17}$, čo je typické pre obrazovú steganografiu. Z tohto dôvodu sa steganografia spustiteľných súborov stala malo populárnou, avšak je stále využiteľná pre vkladanie menších dát, pričom ako uvádza [16], po kombinácií a vylepšení niekoľkých identifikovaných metód je možné dostať sa na kapacitu až $\frac{1}{36}$, čo predstavuje výrazne zlepšenie. [16]

Implementovaná metóda IS bola aj v [2], avšak s rozdielom, že substitúcie boli vykonávané v čase spájania modulov (angl. *link-time*). To implikuje lepšiu mieru detekcie živosti príznakov, čo znamená lepšiu rýchlosť kódovania. Taktiež bol vyvinutý software, ktorý nachádza všetky možné ekvivalentné triedy s nasledujúcimi obmedzeniami:

1. zdrojový a cieľový operand pôvodnej inštrukcie musia byť rovnaké aj v nahradzujúcej inštrukcii,

2. sada okamžitých hodnôt (angl. *immediate value*) bola rozšírená o jej negáciu, pričom sa to týka len najčastejšie sa vyskytujúcich inštrukcií.

Uvedené obmedzenia museli byť zavedené kvôli časovej zložitosti vytvoreného programu, pričom bolo experimentálne overené, že tieto obmedzenia poskytujú len zanedbateľne menšiu kapacitu krycieho súboru. Rýchlosť kódovania metódy IS bola v tomto prípade dosiahnutá $\frac{1}{53}$. [2]

Kódovanie pri použití ekvivalenčných tried

Spolu s vyššie uvedenou metódou bolo v [2] predstavené aj kódovanie s variabilnou dĺžkou. Vhodnosť jeho použitia vychádza zo skutočnosti, že použitie člena ekvivalenčnej triedy závisí od indexu jeho pozície. Výpočet kódovania pre celkový počet n členov ekvivalenčnej triedy sa skladá z nasledujúcich krokov:

1. Ak $\log_2(n) \notin \mathbb{N}$ potom $\log_2(n) = \log_2(n) - 1$. Vždy je preto člen kódovaný aspoň $\log_2(n) - 1$ bitmi.

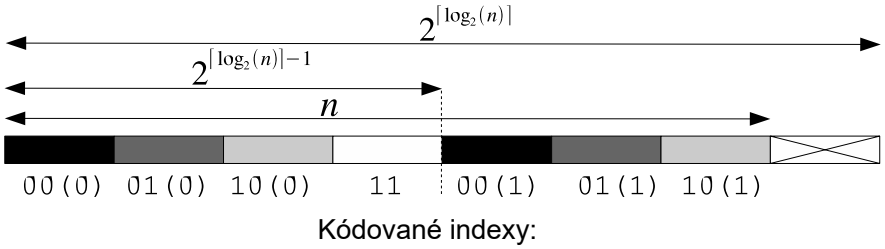
2. Každý zo zostávajúcich $n - 2^{\log_2(n)-1}$ ekvivalentov bude priradený k jednému z $2^{\log_2(n)-1}$ už použitých.

3. Následne sa priradené ekvivalenty od seba odlíšia pridaním tzv. *skupinového bitu*, ktorý definuje skupinu, do ktorej daný ekvivalent patrí.

Priemerná kapacita danej ekvivalenčnej triedy sa potom počíta podľa vzorca 3.1 následovne:

$$b(n) = \lceil \log_2(n) - 1 \rceil + \frac{n - 2^{\lceil \log_2(n)-1 \rceil}}{2^{\lceil \log_2(n)-1 \rceil}} \tag{3.1}$$

Princíp je tohto kódovania je znázornený na obrázku 3.4.



Obr. 3.4: **Kódovanie s variabilnou dĺžkou** – znázornené kódovanie popisuje výsledok vyššie uvedeného algoritmu výpočtu kódovaných indexov ekvivalenčných tried o veľkosti $n = 7$. (Prevzaté a preložené z [2])

3.3.2 Zmena poradia inštrukcií

Metóda *zmeny poradia inštrukcií* (angl. *Instruction Scheduling*) sa typicky vykonáva nad základným blokom (angl. *Basic Block*) strojového kódu. Základný blok je blok inštrukcií s práve jednou vstupnou inštrukciou a jednou výstupnou, pričom sa za každých podmienok musia vykonať všetky inštrukcie bloku v danom poradí. Nesmie byť preto súčasťou základného bloku žiadna skoková inštrukcia s výnimkou poslednej, a ani adresa žiadnej inštrukcie z bloku nesmie byť adresou iného skoku v rámci celého programu. [2] [1]

Blok tvorí typicky $4 - 5$ inštrukcií a je potrebné určiť, ktoré z nich vykonávajú od seba nezávisle operácie, vďaka čomu ich je možné zoraďovať. V tom prípade konkrétna permutácia inštrukcií kóduje bity vkladaných dát. K tomuto cieľu je možné dospieť nasledujúcimi krokmi: [2] [1]

1. Určia sa platné poradia inštrukcií zostrojením grafu závislostí blokových inštrukcií, v ktorých sú závislé inštrukcie spojené orientovanými hranami. Potom cyklickým odstraňovaním inštrukcií z grafu, ktoré nezávisia od iných inštrukcií v grafe je možné určiť platné poradia. Odstránených inštrukcií z grafu v rámci jednej iterácie môže byť viacero. V tomto bode je pripravených viacero sád inštrukcií ktoré je možné permutovať.

2. Vetveným a viazaným algoritmom (angl. *Branch and Bound Algorithm*) na výber inštrukcií z pripravených sád je možné vygenerovať všetky možné permutácie.

Zoraďovanie párov inštrukcie MOV

Myšlienkou tejto techniky je zmena poradia inštrukcií `MOV` v rámci páru, pričom pre tieto dve inštrukcie musia platiť nasledujúce podmienky:

- inštrukcie v rámci páru musia byť umiestnené bezprostredne za sebou,

- inštrukcie musia byť od seba nezávislé,
- nesmie ísť o dve totožne inštrukcie a
- nesmie existovať v rámci celého programu skoková inštrukcia s adresou druhej inštrukcie z páru (došlo by k preskočeniu prvej inštrukcie).

Metóda je absolútne odolná voči štatistickým útokom na binárny súbor a poskytuje vkladanie jedného bitu informácie na jeden pár inštrukcií. Kódovanie môže špecifikovať lexikografické poradie inštrukcií v rámci páru. [33]

3.3.3 Metóda založená na rozložení kódu

Základné bloky vždy tvoria rôzne dlhé neoddeliteľné reťazce blokov, medzi ktorými existuje prechodová cesta. Ak sa však medzi dvoma po sebe nasledujúcimi základnými blokmi nenachádza žiadna prechodová cesta, ide o susediace reťazce blokov a tie je možné od seba oddeliť. Výsledkom môže byť spustiteľný súbor s ľubovoľným poradím reťazcov blokov (zmena poradia funkcií atď.), ktoré kóduje bity vkladaných dát. [2] [1] [16]

Praktické použitie tejto metódy komplikuje spôsob, akým sa moduly zdrojového kódu prekladajú. Jednotlivé moduly sa nikdy neprekladajú naraz, preto prekladač, v čase prekladu, nemá prehľad o celom kóde, čím vzniká program, ktorý obsahuje duplicitný kód. To znamená, že jednotlivé reťazce v programe nemusia byť jedinečné, čo je treba ošetriť pre jednoznačnosť kódovania vkladaných bitov dát. [2] [1]

Pre toto ošetrenie je nutné očíslovať a kvalifikovať všetky reťazce blokov nezávisle od ich polohy v programe. Preto diferenciácia medzi dvoma reťazcami nesmie brať do úvahy žiadne informácie o polohe, kvôli čomu je nutné zanedbať všetky premiestniteľné adresy zakódované v inštrukciách blokov reťazcov. [2] [1]

3.4 Existujúce programové vybavenie

V súčasnosti je voľne dostupné len obmedzené množstvo aplikácií pre steganografiu spustiteľných súborov, na rozdiel od iných druhov steganografií. Menej zdokumentované staršie aplikačné nástroje, ktoré sú použiteľné za účelom ukrytia dát v rámci spustiteľných súborov sú:

- **Clotho**[11] – Ľahko používateľný starší nástroj pre OS Windows, ktorý dokáže skryť citlivé dáta do obrázkov, zvuku, videa, spustiteľných súborov a ďalších rôznych formátov súborov (napr. ZIP, RAR, EXE, DLL, ...). Aplikácia umožňuje šifrovanie skrývaných dát alebo komprimáciu výsledného súboru za účelom zníženia veľkosti. Taktiež je možné skryté dáta extrahovať. Navyše, aplikácia ponúka reštrukturalizáciu dát, aby sa skryté dáta dali extrahovať aj použitím archivačného nástroja WinRAR.
- **StegoStick**[12] – Ide taktiež o starší nástroj, ktorý ponúka možnosť skryť akýkoľvek formát dát do obrázku, zvuku, videa alebo súboru EXE, PDF atď. Podporuje tiež šifrovanie.

Nasledujúce tri aplikačné nástroje sa sústreďujú na inštrukčnú sadu x86 kompatibilnú s architektúrou procesorov IA-32, podobne ako metódy uvedené v predchádzajúcej podkapitole 3.3.

[11] https://www.softpedia.com/get/Security/Encrypting/Nugraha-Clotho.shtml
[12] https://sourceforge.net/projects/stegostick/

Hydan

Hydan[13] je prvým skutočným pokusom – podľa dostupných informácií – o návrh a implementáciu metódy určenej pre steganografiu spustiteľných súborov. Steganografický nástroj implementuje metódu substitúcie funkčne ekvivalentných inštrukcií strojového kódu (vysvetlená v sekcii 3.3.1). Software bol zhodnotený vývojármi jedinou metrikou, a to schopnosťou vkladania bitov za určitý počet bitov krycieho programu. Táto metrika bola ohodnotená hodnotou $\frac{1}{110}$, co implikuje vloženie jedného bitu informácie na každých približne 110 bitov krycieho programu.

ARMaHYDAN

ARMaHYDAN[14] je aplikačný nástroj, ktorý manipuluje s tzv. *voliteľnými bitmi* v inštrukciách procesora ARM. Ide o nedokumentované bity, ktoré nemenia činnosť inštrukcie, avšak zvyčajne sú takto modifikované inštrukcie Disassemblerom dekódované ako nedefinované. Názov tohto software bol odvodený od názvu vyššie spomenutého nástroja Hydan. Tento steganografický nástroj implementuje tzv. *Metódu k šialenstvu* (angl. *Method to the Madness*), ktorú si je možné naštudovať na vyššie uvedenom odkaze.

Avšak, ako poznamenali samotný tvorcovia software, tento spôsob ukrytia dát je príliš zraniteľný a neodporúča pri skrývaní dôležitých dát. Je tomu tak preto, že voliteľné bity inštrukcií sú za štandardných podmienok konzistentné, a preto akákoľvek ich odchýlka je príliš nápadná.

steg86

Novším a modernejším software je *steg86*[15]. Táto steganografická aplikácia je určená pre binárne súbory architektúry x86 a AMD64. Software je schopný skryť tajné dáta do spustiteľného (binárneho) súboru bez ohľadu na formát (ELF, PE, Mach-O, raw, ...). Výhodou nástroja je, že nemá žiaden vplyv na výkon ani veľkosť krycieho súboru. Steganografický nástroj implementuje metódu, ktorá bola predstavená vývojármi nástroja Hydan, pričom sa tvorcovia steg86 odvolávajú na to, že ich objav bol napísaný úplne nezávisle.

Stilo

Asi najmodernejším riešením pre steganografiu spustiteľných súborov je software *Stilo*[16], ktorý posunul celý vedecký výskum tejto steganografie o veľký kus dopredu. Boli v ňom predstavené dve nové techniky popísané v sekcii 3.3.2 a 3.3.3. Taktiež bola implementovaná zlepšená metóda nástroja Hydan. Na záver boli identifikované možné zraniteľnosti týchto metód.

[13] Oficiálna web stránka (http://www.crazyboy.com/hydan/) tohto software a jeho dokumentácie nie je v čase písania tejto práce dostupná.

[14] https://github.com/XlogicX/ARMaHYDAN

[15] https://github.com/woodruffw/steg86

[16] https://www2.cs.arizona.edu/~collberg/Teaching/620/2008/Assignments/tools/stilo/index.html

Kapitola 4

Návrh steganografického nástroja

Cieľom tejto kapitoly je bližší teoretický návrh implementovaného steganografického nástroja pre spustiteľné súbory. Výsledný nástroj bude rozšíriteľným programom pre implementáciu rôznych steganografických metód pre spustiteľné súbory za účelom ich analýzy, zhodnotenia a porovnania.

Podkapitola 4.1 približuje architektúru navrhnutého software, pričom konkrétnejšie popisuje proces vkladania a extrakcie. Podkapitola 4.2 definuje vybranú metódu z existujúceho výskumu a podkapitola 4.3 popisuje jej rozšírenie. Plynule nadväzuje podkapitola 4.4, ktorá predstavuje výsledky môjho skúmania redundancií inštrukčnej sady za účelom modifikácie existujúcej metódy a návrh doposiaľ prakticky nezhodnotenej metódy (sekcia 4.3). Podkapitola 4.5 popíše vhodnosť metrík pre testovanie navrhnutých metód a na záver podkapitola 4.6 spomenie použité technológie.

4.1 Architektúra aplikácie

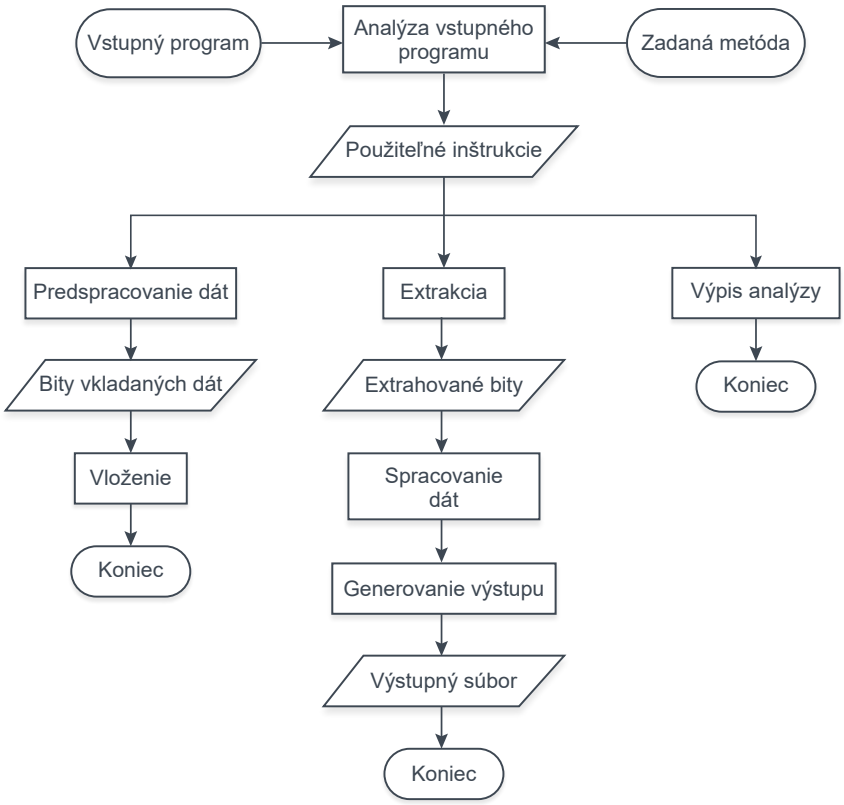
Implementovaný steganografický software bude schopný vkladať a extrahovať rôzne typy dátových objektov (text, videosúbor atď.) podľa zadanej metódy. Keďže ide stále o bajty, pre účely vkladania a extrakcie nezáleží na tom, čo tieto bajty reprezentujú. Avšak pre zlepšenie výstupu extrakcie je vhodné dopredu vedieť význam týchto bajtov, aby mohli byť uložené do súboru a následne správne zobrazené. Z tohto dôvodu bude v rámci dát vkladaná aj prípona súboru, ak sú vkladané dáta definované ako súbor. Ak však pôjde o prostý text zadaný pre vkladanie, spolu s ním sa implicitne vloží prípona textového súboru, ktorá pri extrakcii zabezpečí, že sa požadovaný text zobrazí správne (vo vygenerovanom súbore).

Ako už bolo predstavené vyššie, steganografický software bude prispôsobený pre možnú implementáciu ďalších steganografických metód, avšak jej základom bude implementovaná metóda substitúcie inštrukcií, popísaná v sekcii 3.3.1, s definovanými ekvivalenčnými triedami z [16], ktoré sú bližšie popísané v 4.2. Navyše, táto skupina tried bude rozšírená o ďalšie nájdené redundancie inštrukčnej sady architektúry IA-32, ktoré sú popísané v blogu od P. Kankowskeho [25]. Keďže je tento blog starý niekoľko rokov, redundancie v ňom popísané som experimentálne otestoval v ladiacom prostredí x64dbg[1] pre strojový kód 32- aj 64-bitovej architektúry, aby som zaistil ich aktuálnosť.

Ďalšou implementovanou metódou bude využitie inštrukcií `NOP`. Bližší popis k tomu, akým spôsobom budú tieto inštrukcie kódovať vkladané bity obsahuje sekciu 4.3. Všetky uvedené metódy budú implementované pre formáty spustiteľných súborov ELF (podka-

[1] https://x64dbg.com/

pitola 3.1) a PE (podkapitola 3.2) v 32- aj 64-bitovom režime. Obrázok 4.1 znázorňuje architektúru navrhnutého steganografického nástroja.



Obr. 4.1: **Diagram architektúry steganografického nástroja** – uvedený diagram znázorňuje tok dát steganografického nástroja.

Súčasťou nástroja bude aj konfiguračný súbor, v ktorom budú definované všetky ekvivalenčné triedy substitučnej metódy a zároveň aj ostatné potrebné informácie ďalších implementovaných metód. Podstatou tohto súboru je možnosť zvoliť si jednu z možných variantov kódovaní. To zabezpečí zvýšenú bezpečnosť skrytých dát, pretože tie isté dáta môžu byť zakódované pri rôznych konfiguráciách. Jednou konfiguráciou ekvivalenčnej triedy sa myslí jedno špecifické usporiadanie jej členov, pričom bude použité kódovanie s variabilnou dĺžkou predstavené v časti 3.3.1.

4.1.1 Proces vloženia informácie

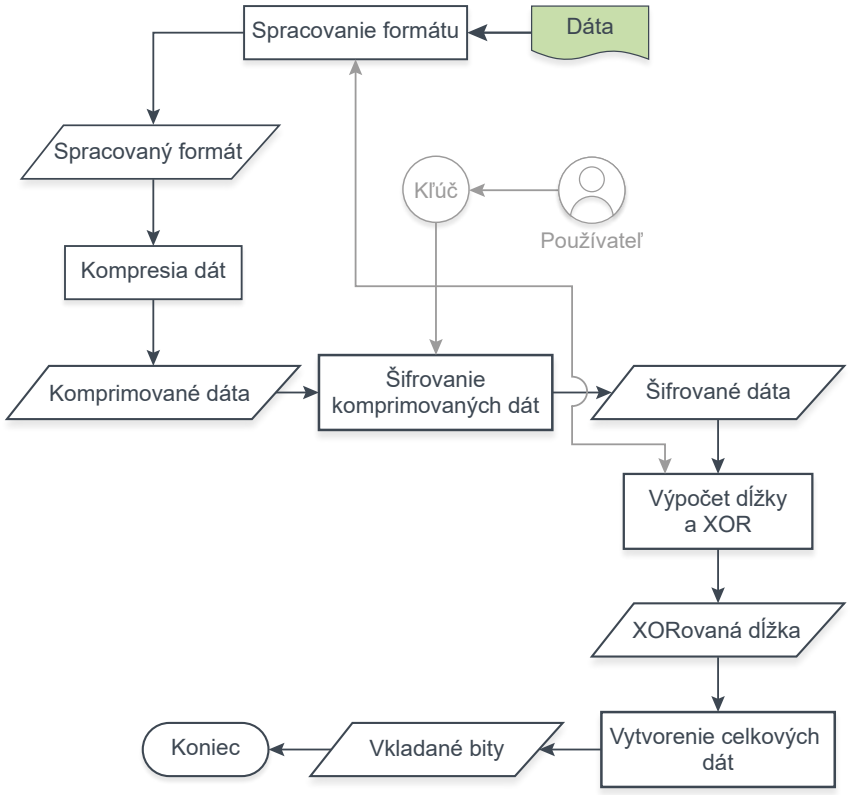
Cieľom vkladania každej steganografickej techniky je vložiť tajné dáta v rámci krycieho súboru s dosiahnutím čo najväčšej kapacity a bezpečnosti. Za účelom naplnenia týchto dvoch kritérií bude vkladaniu predchádzať predspracovanie tajných dát.

V bežnej praxi sa kvôli zníženiu veľkosti súboru používajú rôzne kompresné techniky (sekcia 2.1.7). V prípade implementovanej aplikácie sa budú všetky typy skrývaných dát komprimovať bezstratovou kompresnou technikou, pretože cieľom je spätne extrahovať dáta v presne rovnakej veľkosti a forme. Pri výbere kompresného algoritmu som porovnal všetky bežne dostupné kompresné algoritmy pre prostredia OS Linux a Windows. Nakoniec som

vybral algoritmus LZMA pre jeho najvyšší kompresný pomer aj napriek stredne veľkej časovej zložitosti algoritmu, keďže v prvom rade je cieľom zlepšiť kapacitu krycieho programu.

Pre zaistenie bezpečnosti vložených dát, pre prípad, že by došlo k ich nechcenej detekcii a extrakcii, sa po ich skomprimovaní a tesne pred ich vložením použije šifrovací algoritmus AES. Keďže zámerom je, aby sa kľúč generoval pred vkladaním aj po extrakcii na základe používateľom zadaného hesla, táto symetrická bloková šifra toto umožňuje.

Pre schopnosť neskoršej extrakcie vložených dát je nutné k tajným dátam pripojiť informáciu o tom, aké dlhé dáta sa majú extrahovať. Z tohto dôvodu bude pred tajnú správu vložená jej dĺžka. Opäť, aby nedošlo k nechcenému spozorovaniu použitia steganografie, dĺžka sa pred pripojením k tajným dátam prevedie bitovou operáciou XOR nad zadaným heslom. Následne sa analogicky rovnaká operácia prevedie nad príponou súboru skrývaných dát (viď 4.1) a spolu sa so zakódovanou dĺžkou pripoja k samotnej tajnej správe. Celý výsledný kus vkladaných dát sa nakoniec prevedie na bity (keďže vkladanie bude prebiehať na úrovni jednotlivých bitov), aby s nimi bolo možné ďalej pracovať. Poslednou vstupnou informáciou potrebnou pre vkladanie bude zvolená steganografická metóda, ktorou sa dáta vložia. Celý proces predspracovania vkladaných dát znázorňuje obrázok 4.2.



Obr. 4.2: **Diagram predspracovania vkladaných dát** – uvedený diagram znázorňuje proces predspracovania zadanej tajnej správy určenej pre vkladanie do krycieho programu.

4.1.2 Proces extrakcie informácie

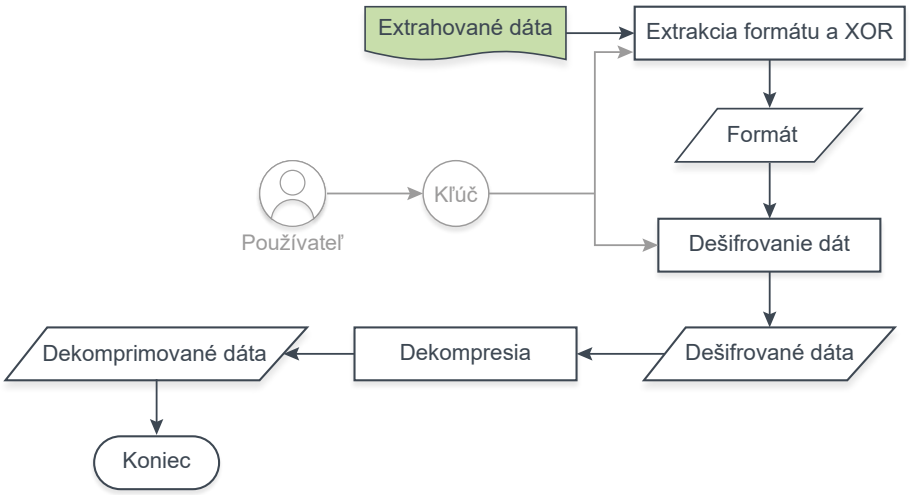
Z podstaty použitia steganografie vyplýva, že dáta vložené do krycieho programu sa od odosielateľa musia dostať k príjemcovi. Ak sa však počas cesty stego-program nestretne s aktívnym útokom, môžeme predpokladať, že je možné dáta zo stego-programu extrahovať.

Pre extrakciu dát je potrebné učiniť presne reverzný postup, ako bol použitý pre vkladanie, preto extrakcia je priamo závislá od vkladania.

Pred extrakciou bude nutné zadať steganografickú metódu zhodnú spolu s tou, pomocou ktorej bolo uskutočnené vkladanie. Proces extrakcie bude rozdelený na dve fázy:

1. extrakcia bitov definujúcich dĺžku zvyšných dát (extrahovaná dĺžka musí byť prevedená bitovou operáciou XOR nad zadaným heslom, ako tomu bolo pri vkladaní),
2. extrakcia zvyšnej časti ukrytých dát.

Keďže bude pred vkladaním učinené predspracovanie tajných dát, po extrakcii bude nutné postupovať obdobne. Najprv sa od získaných dát oddelí pripojený formát skrytej správy danej dĺžky, ktorý sa následne prevedie bitovou operáciou XOR nad zadaným heslom na jeho pôvodnú formu. Zvyšné bity správy budú dešifrované šifrovacím algoritmom za predpokladu, že príjemca správy pozná heslo potrebné pre vygenerovanie kľúča. Ak áno, nasleduje dekompresia dešifrovaných dát, opäť rovnakým kompresným algoritmom ako pri vkladaní. Vytvorí sa požadovaný výstupný súbor a dekomprimované dáta sa doň zapíšu. Výsledkom bude vygenerovaný súbor so správnou reprezentáciou tajnej správy. Celý proces spracovania extrahovaných dát znázorňuje obrázok 4.3.



Obr. 4.3: **Diagram spracovania extrahovaných dát** – uvedený diagram znázorňuje proces následného spracovania extrahovaných dát.

4.2 Popis zvolenej steganografickej metódy

Zvolenou existujúcou metódou je substitúcia inštrukcií s následujúcimi ekvivalenčnými triedami. Kódovanie týmito triedami je uskutočnené použitím toho variantu z ekvivalenčnej triedy, ktorého pozícia kóduje nasledujúce bity potrebné pre vloženie. V prípade, že pri danej ekvivalenčnej triede nie je uvedené inak, je platná pre 32- aj 64-bitové architektúry.

Inštrukcie so zámenou operandov

Ide o skupinu ekvivalenčných tried inštrukcií, ktoré môžu byť, vďaka redundancii inštrukčnej sady procesora, definované dvomi odlišnými operačnými kódmi:

- `OPCODE r/m, r` alebo
- `OPCODE r, r/m`.

Táto redundancia sa týka inštrukcií `MOV`, `ADD`, `ADC`, `SUB`, `SBB`, `AND`, `OR`, `XOR` a `CMP`, čo vytvára až deväť tried o veľkosti dvoch prvkov. Zámenu operandov zabezpečuje prepnutie Smerového bitu (angl. *Direction bit*) nachádzajúceho sa v operačnom kóde inštrukcie na mieste druhého najmenej významného bitu. Prvá z vyššie uvedených možností je definovaná nulovou hodnotou Smerového bitu a naopak. V tomto prípade je pri prepnutí bitu na opačnú hodnotu nutné tiež prehodiť operandy v rámci bajtu `ModR/M` vyskytujúceho sa hneď za operačným kódom inštrukcie. Jedná sa o trojbitové polia `reg/opcode` a `r/m`.

Ekvivalencia inštrukcií TEST, AND a OR

Za istých podmienok je možné dosiahnuť úplne ekvivalentné chovanie troch rôznych inštrukcií, bez ohľadu na kontext programu. Ide o inštrukcie `TEST`, `AND` a `OR`. Tieto podmienky sú splnené, ak jedna z týchto troch inštrukcií operuje len nad jedným totožným registrom. To znamená, že ak je cieľový aj zdrojový operand inštrukcie totožný register (dve pamäťové miesta v rámci jednej inštrukcie byť nemôžu) potom je možné tieto tri inštrukcie ľubovoľne zamieňať. Keďže dve z uvedených inštrukcií sú zároveň členmi predchádzajúcich tried, vzniká jedna ekvivalenčná trieda o veľkosti päť (inštrukcia `TEST r, r/m` neexistuje). V prípade, že inštrukcia patrí do tejto ekvivalenčnej triedy a nahradzuje sa za jej ekvivalent s prehodenými operandmi, nie je potrebné prehadzovať trojbitové polia bajtu `ModR/M`.

Ekvivalencia inštrukcií SUB a XOR

Za presne rovnakých podmienok ako vo vyššie uvedenej triede, je možné dosiahnuť ekvivalenciu inštrukcií `SUB` a `XOR`. Podľa definície týchto inštrukcií, `XOR` nastavuje príznaky `OF` a `CF` vždy na nulu, pričom `SUB` ich nastavuje podľa výsledku. Napriek tomu, nie je v tomto prípade nutné prevádzať detekciu živosti týchto dvoch príznakov, pretože výsledkom oboch operácií je vynulovanie použitého registra. To znamená, že nikdy nemôže dôjsť k pretečeniu. Rozdielom však je, že tieto dve inštrukcie nastavujú príznak `AF` rozdielnym spôsobom. Preto je pred rozhodnutím, či daná inštrukcia patrí do tejto triedy, nutné uskutočniť detekciu živosti tohto príznaku. Ak je príznak mŕtvy, môžeme inštrukcie zameniť. Vzniká ekvivalenčná trieda o veľkosti štyri, keďže opäť obe inštrukcie existujú v oboch formách poradia operandov.

Ekvivalencia súčtu a rozdielu pri negácii operácie a druhého operandu

Táto ekvivalencia vychádza zo základnej matematickej vlastnosti sčítania a odčítania, a teda, že pripočítanie kladného čísla je ekvivalentné k odčítaniu záporného a tiež odčítanie kladného je ekvivalentné k pripočítaniu záporného. Vznikajú tak ďalšie dve ekvivalenčné triedy, obe o veľkosti dva. Na základe definície nastavovania príznakov inštrukcií `ADD` a `SUB` vyplýva, že tieto inštrukcie sú ekvivalentné, len ak sa odlišne nastavené príznaky `OF`, `CF` a `AF` ďalej nepoužívajú. Je preto nutná detekcia živosti všetkých troch príznakov.

4.3 Popis rozšírenia použitej steganografickej metódy

Ako už bolo zmienené v podkapitole 4.1, nasledujúce štyri sekcie definujú redundancie predstavené v blogu od P. Kankowskeho [25], pričom ich aktuálnosť som experimentálne overil.

Inštrukcia TEST s dvomi možnými variantmi

Inštrukcia `TEST` s okamžitou hodnotou na pozícii zdrojového operandu (na pozícii cieľového byť nemôže) môže mať dve rôzne podoby. Rozdielom je hodnota v poli `reg/opcode` v rámci bajtu `ModR/M`. Možné, validné hodnoty sú `0b000` alebo `0b001`. Vzniká ekvivalenčná trieda veľkosti dva.

Prehodenie pamäťových registrov Base a Index

V prípade výskytu pamäťového operandu pri akejkoľvek inštrukcii, pričom tzv. *Scale* pamäťového registra *Index* je jeden, pamäťové registry *Base* a *Index* môžu byť vymenené. Avšak, musia platiť podmienky, že register Base nesmie byť registrom `ESP` akejkoľvek veľkosti, pretože ak by toto nastalo, po ich výmene by sa register `ESP` ocitol na mieste pamäťového registra Index, čo je podľa manuálu Intel [11] zakázané. Taktiež je nutné vynechať inštrukcie obsahujúce register `EBP`, pretože pri jeho použití na mieste pamäťového registra Base dochádza k implicitnému výberu pamäťového segmentu SS pričom vo všetkých ostatných prípadoch je implicitne zvolený segment DS. Túto zámenu je možné prevádzať len v 32-bitovom režime.

Zameniteľnosť inštrukcií SHL a SAL

Inštrukcia aritmetického posunu vľavo (`SAL`) a logického posunu vľavo `SHL` je za každých podmienok ekvivalentná. Táto redundancia bola navrhnutá pravdepodobne pre úplnosť inštrukcií posunov, keďže posuny vpravo sa líšia. Vzniká ďalšia ekvivalenčná trieda o veľkosti dva.

Možnosť voľby operačného kódu inštrukcie v 32-bitovom režime

Inštrukcie `ADD`, `SUB`, `AND`, `OR`, `XOR`, `ADC`, `SBB` a `CMP` majú v 32-bitovom režime, pri použití okamžitej hodnoty veľkosti jeden bajt, dva rôzne varianty operačných kódov. Z podmienky bajtovej okamžitej hodnoty vyplýva, že prvý operand môže byť buď pamäťový, alebo register, avšak ten bezpodmienečne len o veľkosti jedného bajtu (výnimku tvorí register `AL`, pri ktorého použití je operačný kód inštrukcie odlišný). Operačný kód sa opäť líši v Smerovom bite, avšak tentokrát nie je potrebné nič ďalšie zamieňať (ako tomu bolo pri formách inštrukcií s prehodenými stranami operandov). Vzniká tak osem ekvivalenčných tried, každá o veľkosti dva.

Využitie oficiálne dokumentovaných inštrukcií NOP

Inštrukcia `NOP` neprevádza žiadnu operáciu a nemodifikuje ani príznakový register. Tieto inštrukcie dĺžky jeden a viac (oficiálne až deväť) bajtov sa podľa manuálu procesorov Intel [11] využívajú pre pamäťové zarovnanie toku inštrukcií, ktoré zvyšuje rýchlosť spracovania inštrukcií procesorom a nemajú žiaden vplyv na vykonávanie, okrem modifikácie hodnoty registra `EIP` – ukazovateľ aktuálnej inštrukcie na pamäťovom zásobníku.

Chovanie viacbajtových verzií inštrukcie `NOP` je totožné s tou jednobajtovou aj napriek tomu, že používajú pamäťové operandy či registre. V skutočnosti sa však k ani jednému z nich nepristupuje, a to ani v 64-bitovom, ani inom režime. Význam pamäťového operandu je, že umožňuje vytvorenie potrebného počtu bajtov v rámci jednej inštrukcie `NOP`. Oficiálne bajtové sekvencie inštrukcií `NOP`, ktoré sú uvedené v manuáli inštrukčnej sady Intel sú predstavené v tabuľke 4.1.

Tabuľka 4.1: Odporúčané viacbajtové sekvencie inštrukcie `NOP`

Dĺžka	Mnemotechnická pomôcka	Sekvencia bajtov
2	`66 NOP`	`0x66 90`
3	`NOP DWORD ptr [EAX]`	`0x0F 1F 00`
4	`NOP DWORD ptr [EAX + 0x00]`	`0x0F 1F 40 00`
5	`NOP DWORD ptr [EAX + EAX*1 + 0x00]`	`0x0F 1F 44 00 00`
6	`66 NOP DWORD ptr [EAX + EAX*1 + 0x00]`	`0x66 0F 1F 44 00 00`
7	`NOP DWORD ptr [EAX + 0x00000000]`	`0x0F 1F 80 00 00 00 00`
8	`NOP DWORD ptr [EAX + EAX*1 + 0x00000000]`	`0x0F 1F 84 00 00 00 00 00`
9	`66 NOP DWORD ptr [EAX + EAX*1 + 0x00000000]`	`0x66 0F 1F 84 00 00 00 00 00`

Pri experimentovaní s týmito inštrukciami som zistil, že sufix nulových bajtov všetkých inštrukcií z tabuľky, ktorých dĺžka je väčšia ako tri, je možné ľubovoľným spôsobom modifikovať bez toho, aby došlo k zmene chovania inštrukcie `NOP` či dokonca k zmene inštrukcie ako takej. Výnimkou je trojbajtová inštrukcia, ktorej sufix nulových bajtov byť modifikovaný nemôže.

To ma priviedlo k návrhu techniky, ktorá spočíva v tom, že dáta požadované pre vkladanie môžu nahradiť tieto nulové sufixy. Bezpečnosť vložených dát je potom daná bezpečnosťou, ktorú poskytuje predspracovanie vkladaných dát. Veľkou výhodou tohto vkladania je, že viacbajtové inštrukcie `NOP` sú častou súčasťou strojového kódu.

Špeciálnymi prípadmi sú tieto inštrukcie s dĺžkou dva a tri bajty. Druhá z nich hoci obsahuje nulový sufix, no nie je možné ho modifikovať. Preto budú z týchto inštrukcií vytvorené dve ekvivalenčné triedy. Prvá obsahujúca všetky možné kombinácie inštrukcií o celkovej dĺžky dva bajty a druhá, obsahujúca všetky možné kombinácie inštrukcií o celkovej dĺžke bajtovej sekvencie rovnej tromi. Súčasťou týchto tried je aj inštrukcia `FNOP` – analógia `NOP` pre FPU.

4.4 Teoretický návrh ďalších nájdených redundancií

Nasledujúce sekcie definujúce redundancie inštrukčnej sady procesorov kompatibilných s IA-32, poskytujú len teoretické poznatky, ktoré som zistil pri práci a skúmaní ďalších možností popri návrhu vyššie uvedených metód. Tento popis preto môže ďalej slúžiť pri analýze a návrhu konkrétnej metódy pre steganografiu spustiteľných súborov či nejakej jej modifikácie.

Nahradenie inštrukcií NOP za iné alternatívne varianty

Všeobecne by bolo možné vymyslieť nekonečné množstvo ekvivalenčných inštrukcií, ak by nebolo potrebné dodržať celkovú veľkosť binárneho súboru. Ako už bolo uvedené v 3.3.1, táto možnosť je príliš komplikovaná pre implementáciu, preto sa v nasledujúcom príklade obmedzím len na zachovanie bajtovej dĺžky inštrukcií.

Keďže sa klasická jednobajtová inštrukcia `NOP` niekedy disassemblerom strojového kódu reprezentuje ako inštrukcia `XCHG eax, eax` či dvojbajtový `NOP` ako `XCHG ax, ax`, bolo by možné zamieňať každú inštrukciu `NOP` (existujú aj viacbajtové) za rôzne „nič nerobiace“ inštrukcie. V tomto prípade by bolo možné použiť inštrukcie `MOV`, `MOVSXD`, `XCHG` alebo `LEA` tak, aby cieľový aj zdrojový operand boli totožné. Rozšírením kódovania by v tomto prípade mohlo byť použitie týchto inštrukcií nad všetkými možnými registrami. Veľmi priaznivou výhodou týchto inštrukcií zostáva, že žiadna z nich nijako nemodifikuje príznakový register.

Je však nutné dať si pozor pri použití týchto inštrukcií na 64-bitovej architektúre. Ako sa ukázalo, vyššie uvedené inštrukcie nad 32-bitovými registrami nulujú hornú polovicu použitého 64-bitového registra. Z tohto dôvodu sú použiteľné len nasledujúce varianty inštrukcií (uvedené pre stručnosť len nad akumulátorom):

- `MOVSXD rax, eax` – inštrukcia s oboma operandmi rovnakej veľkosti neexistuje, pretože ide o presun s rozšírením znamienka.
- `MOV rax, rax`
- `XCHG rax, rax` – má odlišný kód (`0x4890`) ako inštrukcia `NOP` (`0x90`), avšak niektoré disassemblery túto inštrukciu interpretujú ako `NOP`. Je to pochopiteľné, keďže `0x48` je len prefix k operačnému kódu, vyskytujúci sa pri 64-bitových formách inštrukcií. Oficiálna dokumentovaná dvojbajtová inštrukcia `NOP` má kód `0x6690`.
- `LEA rax, [rax + 0x00]`

Nevýhodou je, že bajtová dĺžka vyššie uvedených inštrukcií je odlišná, preto by bolo nutné nejakým spôsobom zabezpečiť, aby sa ich dĺžky rovnali dĺžke nahradzovanej inštrukcie `NOP`. Jedným z riešení by mohlo byť doplnenie chýbajúcich bajtov jednobajtovými či viacbajtovými inštrukciami `NOP`.

Použitie nedokumentovaných NOP inštrukcií

Súčasťou inštrukčnej sady procesorov sú aj tzv. nedokumentované inštrukcie `NOP`. Ich operačný kód je v rozsahu `0x0f18`–`0x0f1f` a `0x0f0d`. V rámci týchto inštrukcií je možné vkladať požadované bity na určité miesta v rámci bajtu `ModR/M`. Pre využitie tejto redundancie by bolo potrebné urobiť hlbšiu analýzu chovania týchto inštrukcií na rôznych architektúrach procesorov.

Využitie prefixov inštrukcií

Prefixy sú akýmisi modifikátormi použitej inštrukcie. Niektoré majú definované poradie v rámci kódu inštrukcie (napr. REX alebo tzv. povinné prefixy, angl. *Mandatory prefixes*), no iné sa môžu vyskytovať v ľubovoľnom poradí, niekedy dokonca aj vo väčšom počte. Práve táto skutočnosť by mohla byť využitá pre návrh ďalšej steganografickej metódy. Poradie prefixov, ktorých pozície nie sú pevne definované by mohli kódovať bity vkladanej informácie. Je celkom bežnou praxou, že sa konkrétne inštrukcia `NOP` vyskytuje s duplikáciou prefixov `0x66` či `0x2e`. Keďže pridaním prefixu by sa kód inštrukcie zväčšil, bolo by možné pôvodnú inštrukciu nahradiť jej kratším variantom, ktorý by sa takto umelo zväčšil na požadovaný počet bajtov.

4.5 Vyhodnocovanie vlastností steganografických metód

Zhodnotenie vlastností steganografií sa zameriava na otestovanie nepostrehnuteľnosti, robustnosti a kapacity. Avšak steganografia spustiteľných súborov je od ostatných v tomto jedinečná. Jedinou metrikou oficiálne testovanou pri návrhu metód z podkapitoly 3.3 vo výskumoch [2] a [16] bolo zhodnotenie kapacity techník na určitom počte binárnych programov rôznych veľkostí. Kapacita bola meraná metrikou rýchlosti kódovania (angl. *Data Rate*), ktorá definuje počet použitých bitov za určitý počet bitov programového kódu, napr. rýchlosť kódovania metódy substitúcie inštrukcií pri použitých ekvivalenčných triedach z [16] je $\frac{1}{110}$ (každých 110 bitov programového kódu sa zakóduje 1 bit vkladaných dát).

Zhodnotenie nepostrehnuteľnosti sa v dohľadaných vedeckých článkoch prakticky nerealizuje až na niektoré prípady (napr. [6] a [49]), kedy sú popísané testovacie prípady tzv. *Black-box* testov po použití vkladania nad krycím programom. Keďže zhodnotiť ekvivalenciu dvoch programov je mimoriadne náročný proces, zhodnotenie tejto metriky prudko závisí od miery pokrytia testovacích scenárov alebo sa spolieha na teoretickú podstatu implementovanej metódy v prípade neuskutočnenia tohto testovania.

Poslednou vlastnosťou steganografie je robustnosť, ktorá opäť naberá špecifický a odlišný význam v prípade tejto steganografie. Keďže je modifikácia binárneho programu v kontexte testovania robustnosti, ako je tomu pri ostatných druhoch steganografie, neprijateľná (stratová kompresia, dolnopriepustné filtrovanie atď.), jediným experimentom, ktorý môže byť v tomto ohľade uskutočniteľným, je spustenie procesu extrakcie nad bezstratovo komprimovaným spustiteľným súborom. Tento experiment môže byť prevedený skôr z čírej zvedavosti správania sa implementovaného nástroja za týchto podmienok, akoby mal skutočne testovať robustnosť binárneho programu.

4.6 Použité technológie

Implementácia vyššie navrhnutého riešenia bude realizovaná v programovacom/skriptovacom jazyku *Python* verzie 3.9.7 s využitím objektovo orientovaného návrhu (OOP). Python je taktiež prenosný, pričom je natívne podporovaná integrácia s jazykom C, čo sa kvôli rýchlosti implementovaného riešenia vhodne využije (pre prácu na úrovni bitov či bajtov).

Pre získanie základných informácií o poskytnutom binárnom programe bude pre účely ďalšej analýzy využitá utilita prostredia OS Linux, *objdump* – súčasť skupiny programov s názvom *GNU binutils*, ktoré pracujú nad binárnymi súbormi.

Pre disassembling (preklad bajtov binárneho súboru na inštrukcie strojového kódu) bude použitý voľne dostupný nástroj *iced_x86*[2] aktuálnej verzie 1.17.0, programovaný v jazyku Rust s podporou všetkých inštrukcií procesorov Intel a AMD. Ide o niekoľkokrát rýchlejší nástroj v porovnaní s inými dostupnými riešeniami pre rovnaké použitie s podporou pre použitý jazyk Python3. Výhodou jeho použitia je tiež rozhranie, ktoré poskytuje nad dekódovanými inštrukciami spustiteľného súboru.

[2] https://github.com/icedland/iced

Kapitola 5

Implementácia steganografického nástroja pre spustiteľné súbory

Táto kapitola popisuje implementačné detaily navrhnutého software z kapitoly 4. Podkapitola 5.1 definuje základné možnosti spustenia implementovaného software. Podkapitola 5.2 popisuje postup prekladu zadaného binárneho súboru na strojové inštrukcie (angl. *Disassembling*, ďalej bude používaný tento výraz) a podkapitola 5.3 približuje prípravu dát pre analýzu spustiteľného súboru. Kľúčovou časťou nástroja je selekcia inštrukcií vhodných pre aplikáciu steganografie a analýza binárneho súboru (podkapitola 5.4). Kapitola je ukončená popisom princípu vkladania (podkapitola 5.5) a extrakcie (podkapitola 5.6) spolu s nutným spracovaním tajnej informácie.

5.1 Spustenie steganografického nástroja

Steganografický software pre spustiteľné súbory funguje na príkazovom riadku a je možné ho spustiť niekoľkými možnými spôsobmi. Súčasťou každého spustenia musí byť práve jeden z nasledujúcich povinných operandov označujúcich režim:

- `e/embed` – režim označuje vkladanie tajných dát do krycieho programu. Pri spustení je nutné poskytnúť prepínač `-c/--cover-file`, ktorého hodnota predstavuje cestu ku kryciemu programu. Ďalej je potrebné použiť prepínač `-s/--secret-message`. Jeho hodnota môže predstavovať buď cestu k súboru rôzneho formátu, alebo text určený pre vkladanie. Ak je zadaná cesta, vkladaný bude celý obsah súboru, ktorý daná cesta definuje.

- `x/extract` – režim označuje extrakciu skrytých dát zo stego-programu. Je nutné uviesť prepínač `-g/--stego-file`, ktorého hodnota predstavuje cesta k spustiteľnému súboru. Tento súbor sa použije ako stegosúbor a budú z neho extrahované dáta.

- `a/analyze` – režim označuje výpis analýzy krycieho programu. Preto je nutné zadať prepínačom `-c/--cover-file` súbor, ktorého steganografický potenciál sa má zanalyzovať a vypísať.

Pri všetkých vyššie uvedených režimoch je potrebné špecifikovať steganografickú metódu, ktorá sa v danej súvislosti použije. Implementované sú metódy, ktoré označujú nasledujúce hodnoty prepínača `-m/--method`:

- `sub/instruction-substitution` – hodnota definuje použitie substitúcie ekvivalenčných tried definovaných v podkapitole 4.2.
- `ext-sub/extended-substitution` – hodnota definuje použitie vyššie uvedenej substitúcie, ktorá je rozšírená ekvivalenčné triedy definované v podkapitole 4.3.
- `nops/nops-using` – hodnota definuje použitie oficiálne dokumentovaných inštrukcií `NOP`, ktoré sú popísané na konci podkapitoly 4.3.
- `ext-sub-nops` – hodnota definuje použitie kombinácie všetkých troch vyššie uvedených metód.

Navyše je možné nástroj spustiť s prepínačom `-f/--force`, ktorý robí podrobnejšiu analýzu inštrukcií (bude vysvetlené v sekcii 5.4) za účelom nájdenia vhodných miest pre vkladanie dát. Táto funkcia bola od zvyšku oddelená samostatným prepínačom, pretože spôsobuje vysokú časovú zložitosť algoritmu, kvôli čomu beží analýza v radoch niekoľkých minút, avšak na úkor získania vyššej kapacity spustiteľného súboru. V prípade, že tento prepínač použitý pri vkladaní, v dôsledku zhodnej analýzy ho je nutné použiť aj pri extrakcii.

5.2 Disassembling

Po analýze zadaných argumentov príkazového riadku sa nasleduje analýza zadaného binárneho súboru. Trieda `Disassembler` začína zavolaním utility príkazového riadku `objdump` s prepínačom `-h`, ktorý vypíše všetky hlavičky vyskytujúce sa v spustiteľnom súbore. Následne sa tento výstup analyzuje, pričom sa získajú informácie potrebné pre disassembling strojového kódu. Týmito informáciami sú:

- `bitness` – určuje, inštrukcie akej architektúry strojového kódu daný binárny súbor využíva (32- alebo 64-bitovú).
- pole všetkých kódových sekcií `code_sections` (možné disassemblovať na inštrukcie), pričom každá položka poľa je reprezentovaná trojicou dát:
  - `size` – určuje veľkosť danej kódovej sekcie.
  - `vma` (virtuálna adresa sekcie) – ukazovateľ na prvú inštrukciu danej sekcie.
  - `foffset` – poradie bajtu v rámci binárneho súboru, na ktorom sa daná sekcia začína.

Predchádzajúce informácie sú dôležité pre vytvorenie inštancie triedy `Decoder`, ktorá je súčasťou rozhrania používaného nástroja `iced_x86`. V tejto chvíli dochádza k analýze inštrukcií kódových sekcií. Výsledkom je pole všetkých inštrukcií, pričom každá inštrukcia je reprezentovaná inštanciou triedy `MyInstruction`, ktorej atribúty sú:

- `instr` – inštancia triedy `Instruction` z používaného nástroja `iced_x86`. Táto trieda poskytuje široké rozhranie pre rôzne operácie nad získanou inštrukciou.
- `foffset` – číselná hodnota reprezentujúca poradie bajtu v rámci súboru, na ktorom daná inštrukcia začína.
- `ioffset` – index dekódovanej inštrukcie.
- `eq_class` – inštancia triedy `EqClassesProcessor`, ktorá bude predstavená ďalej.

5.3 Príprava potrebných dát pred analýzou inštrukcií

Ešte pred samotnou analýzou, vyššie spomenutá trieda `EqClassesProcessor` jednorázovo vytvára jej vlastné inštancie, a to na základe analýzy konfiguračného súboru. Volaním metódy `prepare_eq_classes()` s dekorátorom `@classmethod` sa začína analýza konfiguračného súboru formátu JSON, v ktorom sú uvedené metadáta, primárne za účelom možnosti voľby kódovania jednotlivých ekvivalenčných tried implementovaných metód. Špecifická konfigurácia súboru implikuje určité nastavenie kódovania.

Konfiguračný súbor obsahuje reťazce názvov metód, ktorých hodnotami sú objekty, ktoré špecifikujú ekvivalenčné triedy danej metódy (poradie ekvivalenčných tried, v rámci konfiguračného súboru, nie je definované). Tieto ekvivalenčné triedy pozostávajú, opäť, z názvu triedy, ktorých hodnotami sú objekty určujúce atribúty danej ekvivalenčnej triedy. Týmito atribútmi vždy sú:

- `Description` – obsahuje popis použitia ekvivalenčnej triedy spolu s vlastnosťami, ktoré obmedzujú jej použitie.
- `Members` – definuje pole členov ekvivalenčnej triedy, ktorých poradie je možné meniť, čím sa zmení kódovanie danej triedy.

Pole ekvivalentov je pri ďalšom rozšírení možné definovať ľubovoľne podľa vôle programátora. V závislosti od ich formátu potom prebieha ich analýza, ktorú v prípade rozšírenia je potrebné implementovať. Zvolené existujúce členy tried sú vo formátoch prijateľných pre rozpoznanie programátorom na prvý pohľad, no zároveň vhodných pre jednoduchú analýzu. Vo špecifických prípadoch je možné toto pole nechať prázdne (napr. ekvivalenčná trieda `>3 Bytes Long NOP` popísaná v sekcii 4.3).

So zámerom rýchlejšieho behu programu sú vytvárané inštancie len tých tried, ktorých sa zvolená metóda týka. Každá z inštancií disponuje nasledujúcimi atribútmi:

- `method_name` – názov metódy, pod ktorou je daná ekvivalenčná trieda definovaná.
- `class_name` – názov ekvivalenčnej triedy.
- `desc` – popis ekvivalenčnej triedy.
- `members` – pole členov ekvivalenčnej triedy.
- `encoded_idxs` – pole kódovaných indexov ekvivalentov z poľa `Members` (dopočítavajú sa neskôr a len v prípade, že nebol zvolený režim `analyze`, pre ktorý nie sú potrebné).
- `avg_cap` – priemerná kapacita, ktorú daná trieda poskytuje (vypočítaná vzorcom 3.1).
- `min_cap` – minimálna kapacita, ktorú daná trieda poskytuje.
- `max_cap` – maximálna kapacita, ktorú daná trieda poskytuje.

Minimálna a maximálna kapacita tried nie je zhodná s priemernou v prípade, že počet členov ekvivalenčnej triedy nie je mocninou čísla dva. Je to spôsobené použitím kódovania s variabilnou dĺžkou (popísané v časť 3.3.1). Keďže až v čase vkladania je možné určiť presnú hodnotu kapacity krycieho súboru, analýza môže poskytnúť len rozsah možnej kapacity. Avšak negatívum takejto nepresnosti nie je veľké, pretože pri krycích súboroch bežnej veľkosti sa zväčša minimálna kapacita líši od maximálnej len o pár stoviek bajtov.

5.4 Selekcia a analýza inštrukcií

Časťou s najväčšou časovou zložitosťou je práve selekcia miest krycieho objektu, ktorých steganografický potenciál je použiteľný. Nad triedou `Selector` sa volá metóda `select()` s dekorátorom `@classmethod`, ktorá dostáva na vstup všetky potrebné dáta získané do tohto momentu. Prechodom cez všetky položky poľa `all_instrs[]` získaného pri disassemblingu binárneho súboru, sa kontroluje, či aktuálna inštrukcia patrí do aspoň jednej z ekvivalenčných tried, ktoré sú definované pre zadanú metódu. Ak áno, daná inštrukcia je vybraná ako potenciálne miesto pre vkladanie/extrakciu. Zároveň sa tejto inštrukcii priradí referencia inštancie triedy `EqClassesProcessor` danej ekvivalenčnej triedy (objekty boli vytvorené pred selekciou – podkapitola 5.3) a cyklicky sa sčítavajú hodnoty priemernej, minimálnej a maximálnej kapacity. Tieto hodnoty (a ďalšie vypisujúce sa na štandardný výstup v režime `analyze`) sa na záver selekcie uložia do atribútov triedy `Analyzer`.

Priorizácia ekvivalenčných tried

Aby bola selekcia vždy jednoznačná, bolo potrebné nastaviť jednotlivým ekvivalenčným triedam, ktorých členy sa niekedy môžu prekrývať, priority. Náležitosť aktuálne kontrolovanej inštrukcie do ekvivalenčnej triedy je kontrolovaná najskôr pre triedy s vyššou prioritou. Priority nasledujúcich ekvivalenčných tried boli nastavené následovne (tieto triedy boli uvedené v podkapitolách 4.2 a 4.3):

- `3 Bytes Long NOP` > `2 Bytes Long NOP`
- `Swap base-index registers 32-bit` > `MOV`, `ADD negated`, `SUB negated`
- `TEST/AND/OR` > `AND`, `OR`
- `SUB/XOR` > `SUB`, `XOR`

Určenie priorít vyššie uvedených ekvivalenčných tried bolo nastavené podľa nasledujúcich kritérií:

1. **veľkosť ekvivalenčnej triedy** – zlepšuje kapacitu krycieho objektu, vďaka schopnosti kódovať vyšší počet bitov jedným členom triedy.
2. **bezpečnosť ekvivalenčnej triedy** – aj keď je bezpečnosť použitia danej ekvivalenčnej triedy veľmi dôležitá, v tomto prípade nebola priorizovaná, pretože implementovaný nástroj bude použitý najmä na porovnávanie dátovej rýchlosti (ohľad na bezpečnosť metód a ich zhodnotenie môže byť ďalším rozšírením nástroja v budúcnosti).

Detekcia živosti príznakov

V prípade ekvivalenčných tried `SUB/XOR`, `ADD negated` a `SUB negated` bolo nutné uskutočniť detekciu živosti odlišne modifikovaných príznakov členmi danej ekvivalenčnej triedy. Túto detekciu implementuje statická metóda `__liveness_flags_detection()`. Metóda v nekonečnom cyklu prechádza všetky nasledujúce inštrukcie od aktuálnej, pričom je rešpektovaný tok vykonávania programu. Zo skupiny kontrolujúcich príznakov sa určitý príznak vyradí v prípade, že nastane aspoň jedna z nasledujúcich podmienok:

1. nájde sa inštrukcia, ktorá daný príznak modifikuje skôr ako inštrukcia, ktorá daný príznak testuje;

2. nájde sa inštrukcia `ret`, ktorá definuje koniec aktuálnej funkcie.

Po vyradení všetkých kontrolovaných príznakov sa daná inštrukcia môže nahradiť. Ak sa však pri kontrole niektorého z kontrolovaných príznakov ukáže, že prvá vyššie uvedená podmienka platiť nemôže (nastane detekcia inštrukcie, ktorá testuje príznak), inštrukcia, ktorej steganografický potenciál sa testuje, nemôže byť využitá.

Vyššie uvedená kontrola sa v prípade detekcie akéhokoľvek skoku, ktorým nie je návrat z funkcie (`ret`), končí neúspechom. Avšak existuje aj možnosť testovať príznaky benevolentnejšie, kedy kontrola podmienok pokračuje po vykonávaní toku aj s prevedením nepodmienených skokov. Táto možnosť sa zapína pri spúšťaní programu definovaním prepínača `-f/--force` a ako už bolo povedané v podkapitole 5.1, na úkor väčšej časovej zložitosti môže zlepšiť kapacitu krycieho programu.

5.5 Predspracovanie dát a vkladanie

Celý proces vkladania, vrátane predspracovania dát je implementovaný v triede `Embedder`. Najprv sú statickou metódou `get_secret_data()` získané dáta potrebne pre vkladanie vo forme bajtov. Ak boli dáta načítané zo súboru, táto metóda metóda vracia spolu s nimi aj extrahovanú príponu súboru vkladaných dát, opäť prevedenú na bajty. Ak sa však vkladá len textový reťazec, implicitne je vrátená prípona TXT. Dĺžka prípony je v tejto chvíli obmedzená na veľkosť osem bajtov, čo sa však v budúcnosti môže jednoducho zmeniť.

Predspracovanie vkladaných dát

Súčasťou predspracovania dát pred ich vložením je kompresia. Pre tento proces bola využitá trieda `LZMACompressor` zo štandardného modulu `lzma` jazyka Python3. Pred kompresiou sa nastaví formát `xz`, do ktorého sa majú dáta komprimovať a hodnota `preset`, ktorá definuje mieru snahy o kompresný pomer na úkor časovej zložitosti. V tomto prípade bola uprednostnená kapacita krycieho programu, a preto bola hodnota `preset` nastavená na nulu – kompresný algoritmus LZMA vracia čo najmenšie dáta.

Nasleduje šifrovanie komprimovaných dát s využitím symetrického šifrovania z triedy `cryptography.Fernet`. Táto trieda ponúka rozhranie, ktoré je možné využiť pre generovanie kľúča symetrickej kryptografie zo zadaného hesla používateľom. Aby bolo zaistené, že sa kľúč pre dané heslo vygeneruje stále rovnaký (najskôr v režime `embed`, neskôr v režime `extract`), parameter `salt` funkcie `PBKDF2HMAC()` musí byť nastavený na konštantnú hodnotu o dĺžke 16 bajtov. Táto hodnota bola vygenerovaná funkciou `os.urandom(16)`.

Funkcia `PBKDF2HMAC()` generujúca kľúč odvodeného z hesla, bola inicializovaná hešovacím algoritmom SHA256 pre kontrolu integrity dát. Algoritmus SHA256 je spustený na odporúčaných 390 000 iterácií, čo znižuje zraniteľnosť vytvoreného hesla voči útokom hrubou silou (angl. *Brute Force*). Tento proces je vykonávaný funkciou `gen_key_from_passwd()` implementovanou v module `common.py`, ktorý obsahuje funkcie využívané viacerými implementovanými modulmi.

Ďalej je vypočítaná dĺžka šifrovaných dát (momentálne 32 bajtov), ktorá sa takisto zabezpečí. Tentokrát však operáciou XOR nad zadaným heslom (ak heslo zadané nie je, nie je ani zabezpečená dĺžka dát). Presne rovnakým spôsobom sa zabezpečia bajty definujúce príponu a následne sa z predspracovaných dát vytvorí jeden celok v poradí: dĺžka + prípona + dáta. Celkové dáta sa prevedú z bajtov na bity, a to využitím externého modulu `bitarray`. Zvolený modul vhodne a efektívne implementuje prácu s bitmi, ktoré sú interpretované

poľom pravdivostných hodnôt `True` a `False`. Bity celkových dát sú reprezentované poradím *little-endian*.

Vkladanie dát

Pre overenie, či je možné zadané dáta do krycieho súboru vložiť, uskutoční sa kontrola dĺžky predspracovaných dát a minimálnej kapacity krycieho súboru. Ak by sa vkladané dáta do krycieho súboru nevošli, program sa ukončí s chybovým hlásením. Toto chovanie je nutnosťou a je spôsobené overením integrity dešifrovaných dát po extrakcii. Dáta ktoré sú neúplné, porušujú integritu pôvodných dát, a preto by nemohli byť dešifrované.

Proces vkladania je v súčasnosti obmedzený len na využitie inštrukcií, ktorých operačný kód spĺňuje kódovanie *Legacy*. Iné kódovania operačných kódov inštrukcií (*VEX/XOP* či *3DNow!*) podporované nie sú.

Zložitosť procesu vkladania závisí od príslušnosti danej inštrukcie do ekvivalenčnej triedy. Vkladanie prebieha iteráciou nad objektmi triedy `MyInstruction`, ktoré sú výsledkom selekcie (popísané v podkapitole 5.4). Každou iteráciou sa vykonáva nasledujúci postup:

1. kontroluje sa príslušnosť inštrukcie do ekvivalenčnej triedy,
2. rozhodne sa o vkladanom počte bitov nájdením správneho zakódovaného indexu (metóda `__find_encoded_idx()` popísaná nižšie),
3. vykoná sa postup vkladania typický pre príslušnú triedu,
4. odstráni sa vložený počet bitov zo začiatku bitového poľa `bitarray`.

Veľmi významnou súčasťou algoritmu pre vkladanie dát do krycieho súboru je statická metóda `__find_encoded_idx()`. Postup tejto metódy je nasledujúci:

1. Na základe počtu členov ekvivalenčnej triedy, nad ktorou je metóda volaná, sa vypočíta počet bitov, ktorý daná ekvivalenčná trieda dokáže kódovať. V prípade, že je počet členov mocninou dvojky, počet bitov, ktorý trieda dokáže kódovať je konštantný, inak je v rozsahu ± 1.
2. V prípade, že je maximálny počet bitov väčší ako dĺžka bitov zostávajúcich pre vkladanie, pridajú sa k dátam pre vloženie nulové bity ako výplň (angl. *padding*) na koniec bitového poľa v reprezentácii *little-endian*.
3. Skúsi sa mapovanie vypočítanej dĺžky bitov zo začiatku bitového poľa vkladaných dát na niektorý kódovaný index členov ekvivalenčnej triedy. Ak je mapovanie úspešné, metóda vracia index v poli kódovaných indexov (odpovedá indexu člena ekvivalenčnej triedy, ktorý sa má použiť).
4. Ak sa predchádzajúce mapovanie nepodarilo, skúsi sa mapovanie vypočítanej dĺžky bitov zmenšenej o jeden. V tomto prípade sa už mapovanie podariť musí a vracia sa opäť index v poli kódovaných indexov, ktorý odpovedá indexu člena ekvivalenčnej triedy, ktorý sa má použiť.

5.6 Proces extrakcie a následné spracovanie dát

Proces extrakcie prebieha, rovnako ako vkladanie, cyklicky nad poľom inštrukcií, získaných rovnakou analýzou ako pred procesom vkladania. Podstatou správnej extrakcie dát je poznať dĺžku skrytej informácie. Tá bola pri vkladaní vložená na začiatok vkladaných dát s pevne určenou veľkosťou 32 bajtov. Proces extrakcie začína extrahovaním práve týchto bajtov. Keďže sa pred vkladaním uskutočnilo predspracovanie dát, je potrebné v tejto fáze extrakcie vrátiť tvar prvých 32 bajtov to pôvodnej podoby. V tejto chvíli sa očakáva na vstupe heslo od používateľa, aby sa dĺžka dát mohla (bitovou operáciou XOR nad zadaným heslom) previesť do pôvodnej podoby. Dĺžka dát je podľa zadaného hesla dekódovaná a pokračuje sa cyklickým extrahovaním daného počtu bitov.

Proces extrakcie je viditeľne jednoduchším procesom ako vkladanie. Je však dôležité vypichnúť statickú metódu `__decode()` definovanú v triede `Extractor`. Po prečítaní kľúčových bitov z inštrukcie, na základe jej príslušnosti do ekvivalenčnej triedy, je potrebné zistiť ich index v rámci tejto triedy. To je úlohou metódy `__decode()`, ktorá len jednoducho mapuje extrahované bity na jeden z členov danej ekvivalenčnej triedy. Výsledkom tohto mapovania sú bity jeho kódovaného indexu.

Spracovanie extrahovaných dát

Po zostavení celého poľa extrahovaných bitov prichádza k ich spracovaniu. Spracovanie extrahovaných bitov definuje reverzný proces predspracovania dát pred ich vložením. Najprv sa z celkových bitov extrahuje počet bitov definujúcich príponu formátu vkladaných dát. Tá sa prevedie bitovou operáciou XOR na skutočný tvar prípony, ktorú je možné použiť ako sufix názvu výsledného súboru. Zvyšok dát sa dešifruje rovnakým šifrovacím algoritmom ako pri vkladaní.

Proces dešifrovania spúšťa metóda `decrypt_data()`, pričom sa použije predtým zadané heslo. Ak bolo zadané heslo správne, dáta sú správne dekódované. Avšak v prípade, že je zadané heslo odlišné od toho, ktoré bolo zadané pri vkladaní, trieda `Fernet` vracia výnimku `InvalidToken`, pretože sa vďaka hešovaciemu algoritmu SHA256 zistilo porušenie integrity extrahovaných dát. Táto výnimka je metódou `decrypt_data()` odchytená a program končí s chybovým hlásením o zle zadanom hesle.

Kapitola 6

Testovanie a experimenty nad implementovanými metódami

Pre testovanie všetkých nasledujúcich scenárov bol vytvorený testovací skript `test.py` v jazyku Python3, ktorý pracuje v zložke `/tests`. Povinnou štruktúrou zložky sú ďalšie zložky `data` a `executables`. Význam prvej zložky je, že sa v nej nachádzajú všetky vkladané dáta testovacím skriptom. Druhá slúži pre umiestnenie krycích programov, resp. stegoprogramov, nad ktorými dané testy prebiehajú. Výsledkom všetkých testovacích scenárov je graf s nameranými kľúčovými dátami, ktorý je generovaný knižnicou `matplotlib` verzie 3.5.2. Porovnávať dáta pred vložením a po extrakcii nie je potrebné, pretože, ako už bolo vysvetlené v 5.6, trieda `Fernet`, kontroluje integritu extrahovaných dát.

6.1 Testovanie nepostrehnuteľnosti metód

Keďže zo samotnej analýzy použitých metód a redundancií vyplýva, že ekvivalencia programu pred vkladaním a po ňom, zostáva zachovaná, testovanie tejto vlastnosti bolo prevedené len nad utilitou `hexdump` z prostredia Linux. Prevedenie testovania len nad jednou utilitou je taktiež podmienené časovou náročnosťou vytvorenia *Black-box* testovania nad viacerými programami.

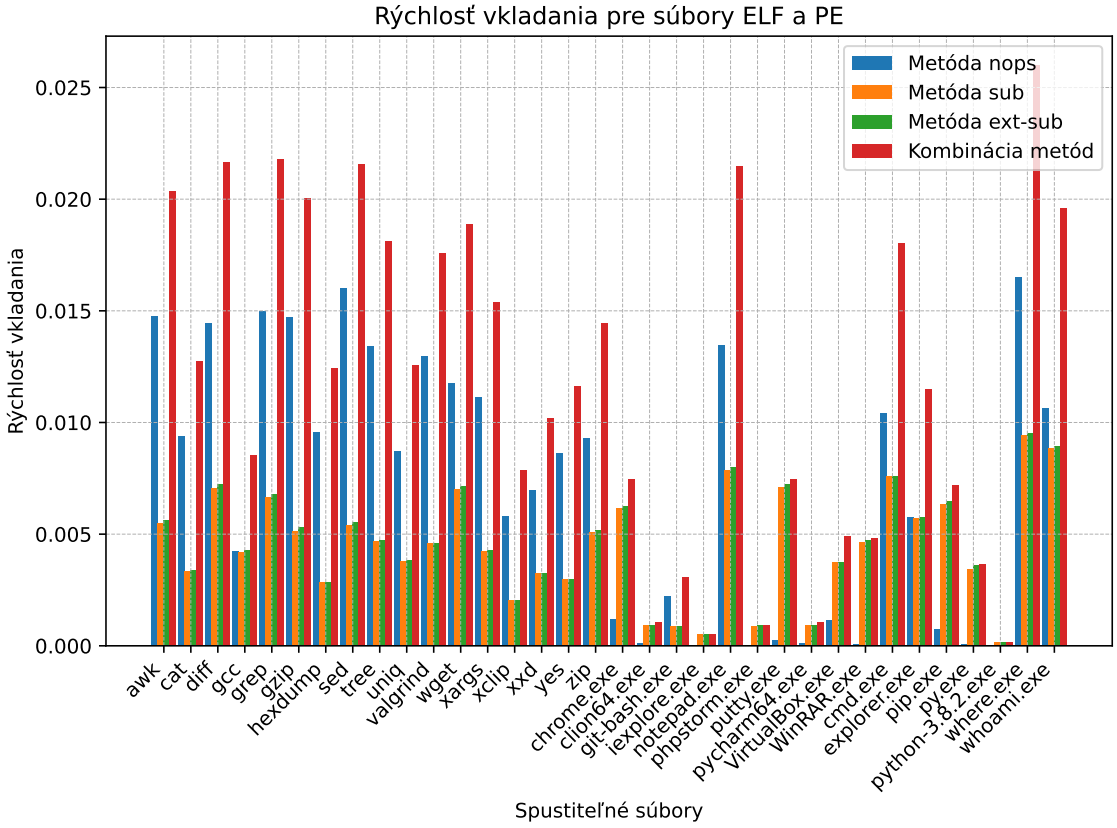
Funkcia `run_check()` najprv vyhľadá utilitu `hexdump` bežnú pre prostredia Linux. Následne je kópia binárneho súboru presunutá do zložky `/tests/executables`. Spustí sa implementovaný steganografický software v nasledujúcom tvare:

```
python3 embed -m ext-sub-nops ../src/main.py -c ./executables/hexdump
                                                    -s ./data/1k.png
```

Tento scenár bol prevedený nad kombináciou všetkých implementovaných metód, pretože sú týmto pokryté všetky z nich. Vyššie uvedené vkladané dáta sú len ukážkou. Nasleduje skupina testovacích scenárov, ktoré otestujú utilitu `hexdump` nad prepínačmi `-b | -c | -C | -d | -n 42 | -o | -s 42 | -v | -x`. Tieto scenáre sa spustia najprv nad pôvodným binárnym súborom, neskôr nad modifikovaným v zložke `./executables` a porovnajú sa výsledky. Verifikácia bola úspešná, čo dokazuje ekvivalenciu binárneho súboru pred a po aplikovaní zvolených steganografických metód.

6.2 Porovnanie dátovej rýchlosti jednotlivých metód

Najpoužívanejšou metrikou pri testovaní metód steganografie pre spustiteľné súbory je dátová rýchlosť (angl. *Data Rate*). Zhodnotenie tejto metriky bolo realizované najmä pre porovnanie dosiahnutých výsledkov implementovaných metód s existujúcimi meraniami. Graf uvedený na obrázku 6.1 znázorňuje dosiahnuté hodnoty pre všetky implementované varianty metód. Pre účely tohto merania bol použitý len základný režim bez prepínača `-f/--force`.



Obr. 6.1: **Graf nameraných dátových rýchlostí** – uvedený graf reprezentuje namerané hodnoty testovacím skriptom `/tests/test.py` (funkcia `test_encoding_rates()`).

Ako je možné vidieť z hodnôt na ose x, použité boli spustiteľné súbory formátov ELF aj PE, pričom sa medzi nimi vyskytujú aj 32-, aj 64-bitové verzie. Tabuľka 6.1 obsahuje zhrnutie priemerných, minimálnych a maximálnych hodnôt pre jednotlivé metódy.

Tabuľka 6.1: Namerané hodnoty dátových rýchlostí

Metóda	Priemerná dátová rýchlosť	Minimálna dátová rýchlosť	Maximálna dátová rýchlosť
`nops`	1/136	0	1/61
`sub`	1/223	1/7143	1/106
`ext-sub`	1/219	1/7143	1/105
`ext-sub-nops`	1/84	1/7143	1/38

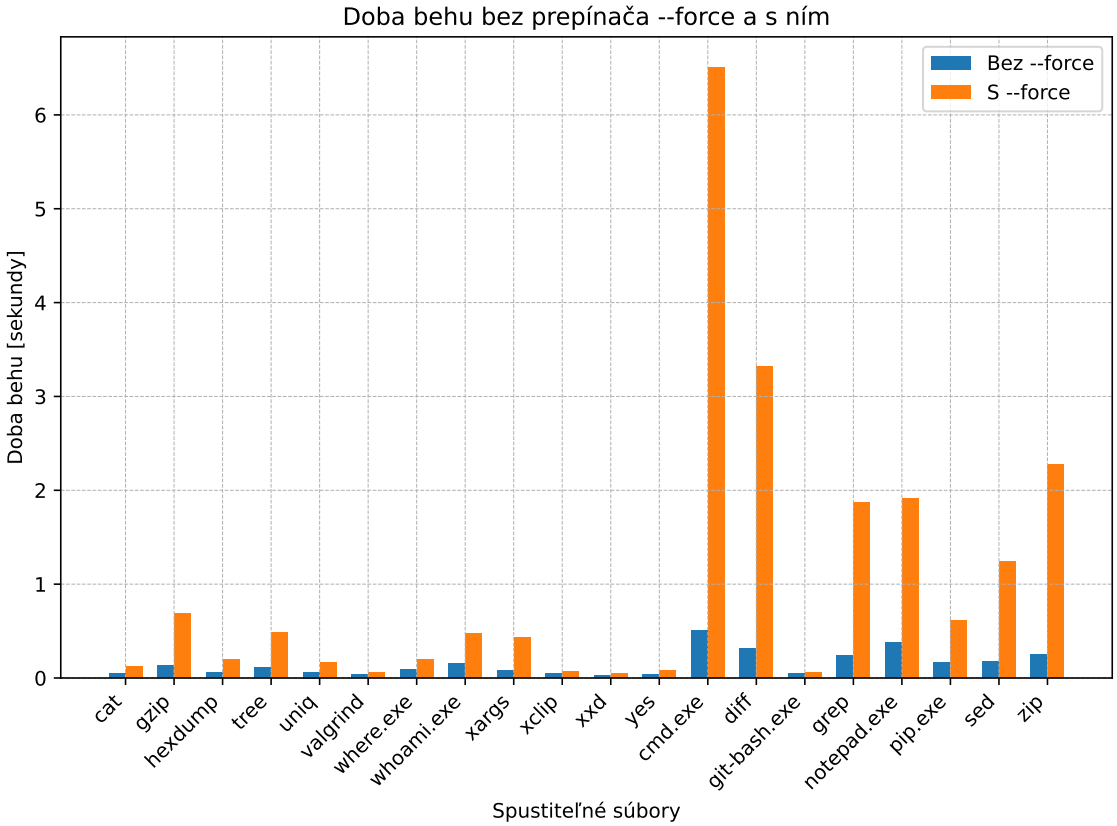
Priemerná hodnota metódy `sub`, ktorá implementuje ekvivalenčné triedy z [16], je viac ako dvakrát nižšia, čo môže byť spôsobené viacerými faktormi. Keďže sa jedná o starší

zdroj, môže ísť o rozdiel spôsobený používanými prekladačmi, či len konkrétnou vzorkou krycích programov.

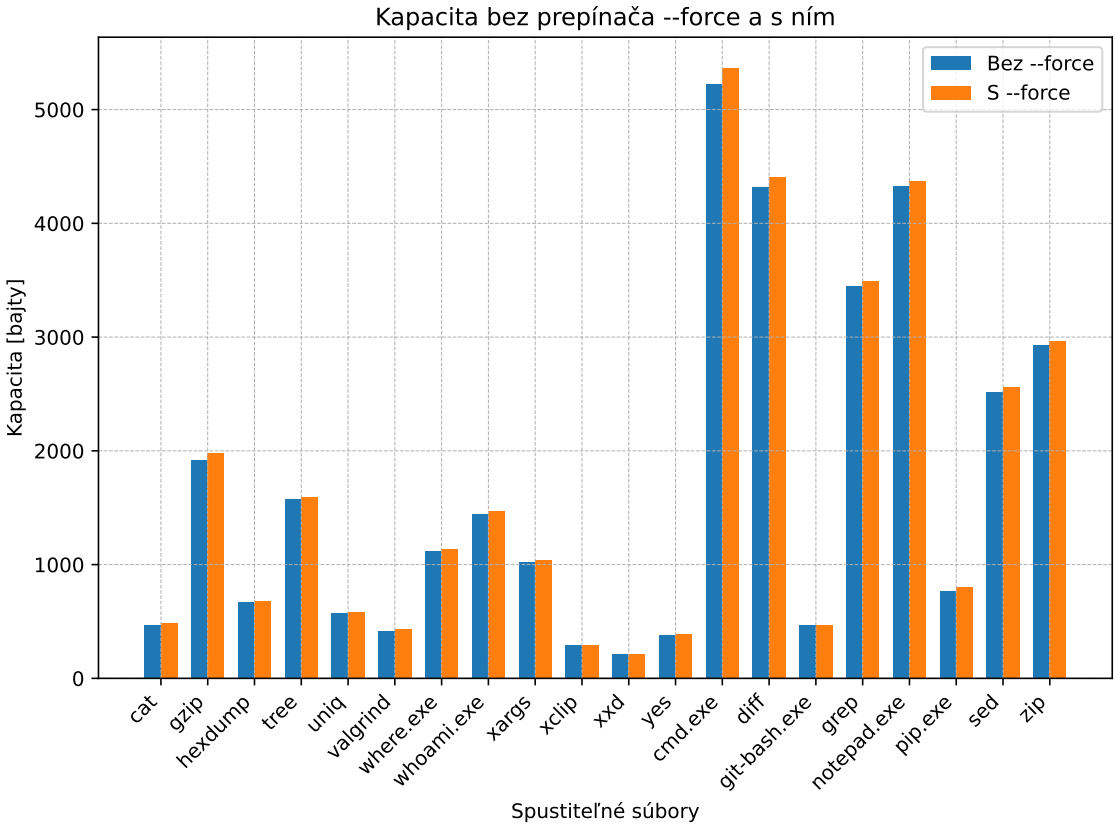
V prípade výsledkov z [2], kde bola dosiahnutá priemerná hodnota metódy substitúcie inštrukcií $\frac{1}{53}$, je možné konštatovať, že mnou dosiahnutá hodnota je len o viac ako $1,5$-krát nižšia od kombinácie metód `ext-sub-nops`. Tento rozdiel je veľmi pravdepodobne spôsobený tým, že mnou použité ekvivalenčné triedy boli hľadané ručne, pričom výsledky z [2] sú vyhodnotené pri použití ekvivalenčných tried nájdených špecializovaným software. Taktiež je zaujímavé si všimnúť dosiahnutú maximálnu hodnotu kombinácie metód, ktorá je vyššia ako priemerná z [2].

6.3 Porovnanie časovej zložitosti a kapacity

Toto porovnanie sa týka analýzy časovej zložitosti a kapacity pri spustení s a bez prepínača `-f/--force`. Prepínač slúži na detailnejšiu kontrolu možnosti substitúcie danej inštrukcie pri kontrole odlišne menených príznakov. Takáto podrobnejšia kontrola sa prevádza trasovaním toku vykonávania programu aj prevádzaním nepodmienených skokov. V základnom režime, bez tohto prepínača, sa pri strete s nepodmieneným skokom rozhodne, že daná inštrukcia byť substituovaná nemôže. Z tohto dôvodu sa očakáva vyššia kapacita dát, avšak na úkor väčšej časovej zložitosti. Obrázky 6.2 a 6.3 vykresľujú namerané hodnoty dĺžky doby behu programu a kapacitu krycieho programu pri oboch režimoch.



Obr. 6.2: **Graf zobrazujúci doby behov pri (ne)použití prepínača `-f/--force`** – tento graf bol vygenerovaný testovacím skriptom, konkrétne funkciou `test_time_force()`.

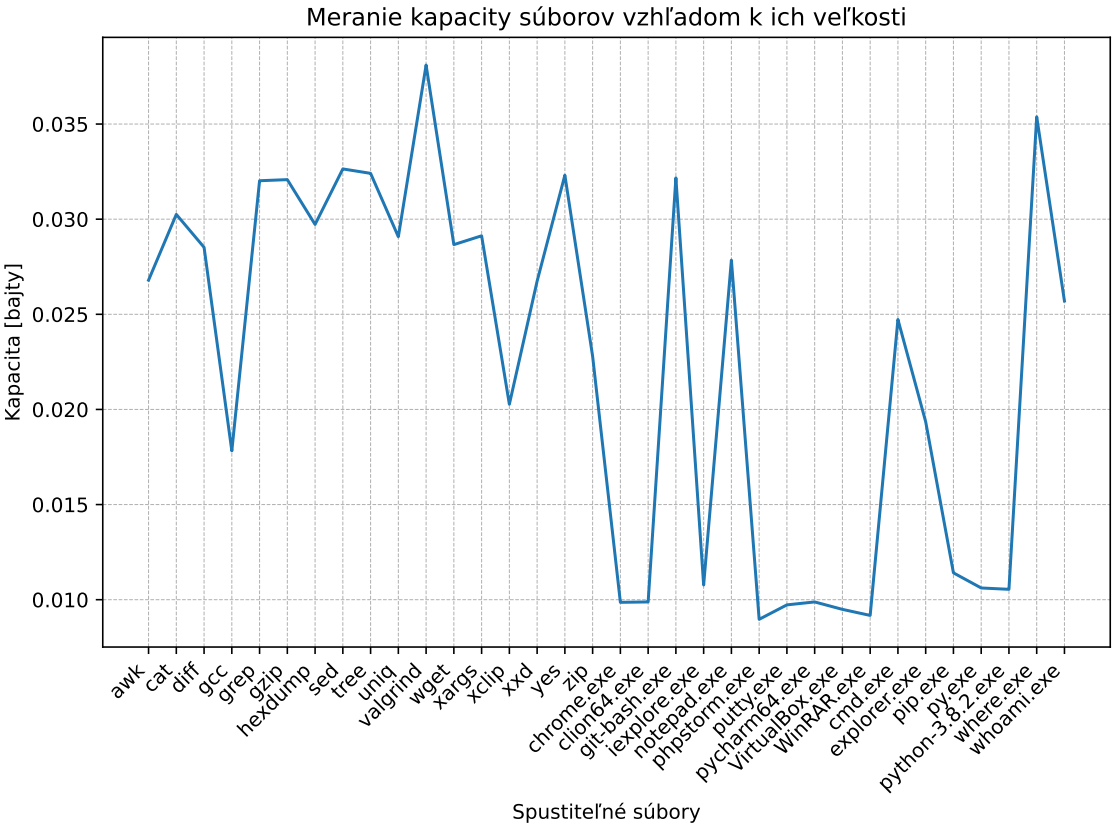


Obr. 6.3: **Graf znázorňuje získanú kapacitu z krycieho programu v oboch režimoch** – tento graf je výsledkom funkcie `test_cap_force()` testovacieho skriptu.

Výsledky testovania ukazujú, že kým je získaná kapacita v režime `force` lepšia v priemere len o $1,9\,\%$, časová zložitosť implementovaného software sa zväčší až o $362\,\%$. Pre testovanie boli použité binárne súbory formátu ELF a PE, rôznych architektúr (32- aj 64-bitové).

6.4 Porovnanie kapacity a veľkosti súborov

Je všeobecným predpokladom, že kapacita spustiteľných súborov z pohľadu steganografie je nízka. Účelom tohto testu bolo overiť uvedenú hypotézu a zobraziť pomer použiteľných bajtov vzhľadom k celkovej veľkosti programového kódu – ide o veľkosť časti binárneho súboru, ktorú je možné preložiť na strojové inštrukcie. Obrázok 6.4 zobrazuje tento pomer pre vybrané spustiteľné súbory formátov ELF a PE pre 32- a 64-bitové architektúry. Pre získanie dát bola analýza spustená pri zadaní kombinácie implementovaných metód (`ext-sub-nops`).



Obr. 6.4: **Graf znázorňuje pomer kapacity ku celkovej veľkosti súboru** – tento graf je výsledkom funkcie `test_capacity()` testovacieho skriptu.

Kapitola 7

Záver

Výsledkom tejto práce je rozšíriteľný software implementujúci existujúcu steganografickú metódu pre spustiteľné súbory, jej rozšírenie, modifikáciu a tiež ich spoločnú kombináciu. Boli preskúmané všetky dostupné možnosti doterajšieho prístupu testovania a hodnotenia metód tejto steganografie. Na základe veľmi obmedzeného množstva získaných vedomostí z tejto časti, bolo navrhnuté vykonané testovanie tak, aby čo najlepšie reflektovalo hlavné metriky steganografie. Taktiež došlo k porovnaniu dosiahnutých hodnôt s existujúcim množstvom zdrojov. V prípade existujúcej metódy je možné tvrdiť, že dosiahnuté hodnoty sa približujú referenčným a v prípade kombinácie implementovaných metód bol dosiahnutý až dvojnásobne lepší výsledok, avšak o niečo menší, aký bol dosiahnutý odlišným prístupom v druhom existujúcom riešení.

Súčasťou práce bolo aj dôkladné štúdium tejto oblasti informačnej bezpečnosti s vysokým potenciálom aj v budúcnosti. Pretože čím ďalej v čase ideme, tým komplikovanejší svet ľudia tvoria, čím vznikajú stále ďalšie možnosti ako ukryť súkromné informácie tak, aby o ich existencii vedeli len tí, ktorým je to dovolené. Steganografia zvyšuje súkromie komunikácie, pretože najsúkromnejšia komunikácia je taká, ktorá nikdy neexistovala.

Vzniknuté riešenie tejto práce ponúka hneď niekoľko vylepšení. Prvým je rozšírenie ponúknutého teoretického návrhu ďalších redundancií inštrukčnej sady, ich implementácia a zhodnotenie. Taktiež, zdrojový kód implementovaného software obsahuje prichystané niektoré kľúčové funkcie pre implementáciu metódy zoraďovania párových inštrukcií MOV (*MOV Scheduling*), čo môže predstavovať ďalšie rozšírenie. Samozrejmý je návrh a implementácia úplne nových steganografických metód pre spustiteľné súbory, keďže vzhľadom na veľkosť dnes používanej inštrukčnej sady je v nej stále možné nájsť mnoho redundancií. Zaujímavou myšlienkou by mohla byť hĺbková analýza skúmaných formátov binárnych súborov s cieľom nájsť využiteľné miesta pre ukrytie dát. V neposlednom rade sa doterajšia práca nevenuje kritériu bezpečnosti implementovaných metód ani z pohľadu testovania, ani ich zhodnotenia, preto by bolo vhodné túto metriku zaradiť medzi ostatné zohľadnené.

Literatúra

[1] Anckaert, B., Sutter, B. D. a Bosschere, K. D. *Steganography for Executables* [online]. Ghent, Belgium: Ghent University, Electronics and Information Systems Department, 2004 [cit. 2022-03-11]. Dostupné z: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.67.8632&rank=1&q=Steganography%20for%20Executables&osm=&ossid=.

[2] Anckaert, B., Sutter, B. D., Chanet, D. et al. Steganography for Executables and Code Transformation Signatures. In: Park, C.-s. a Chee, S., ed. *Information Security and Cryptology – ICISC 2004* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, sv. 3506, s. 425–439 [cit. 2022-03-11]. ISBN 978-3-540-32083-8. Dostupné z: https://link.springer.com/chapter/10.1007/11496618_31.

[3] Anderson, R. a Petitcolas, F. On the limits of steganography. *IEEE Journal on Selected Areas in Communications* [online]. IEEE. Máj 1998, zv. 16, č. 4, s. 474–481, [cit. 2022-01-24]. DOI: 10.1109/49.668971. ISSN 1558-0008. Dostupné z: https://ieeexplore-ieee-org.ezproxy.lib.vutbr.cz/document/668971.

[4] Anees, A., Siddiqui, A. M., Ahmed, J. et al. A technique for digital steganography using chaotic maps. *Nonlinear Dynamics* [online]. Springer. Marec 2014, zv. 75, č. 4, s. 807–816, [cit. 2022-01-27]. DOI: 10.1007/s11071-013-1105-3. ISSN 1573-269X. Dostupné z: https://link.springer.com/article/10.1007/s11071-013-1105-3.

[5] Antonio, H., Prasad, P. W. C. a Alsadoon, A. Implementation of cryptography in steganography for enhanced security. *Multimedia Tools and Applications* [online]. Springer. Máj 2019, zv. 78, č. 23, s. 32721—-32734, [cit. 2022-01-26]. DOI: 10.1007/s11042-019-7559-7. ISSN 1573-7721. Dostupné z: https://link.springer.com/article/10.1007/s11042-019-7559-7.

[6] Aos, A., Naji, A. W., Hameed, S. A. et al. Approved Undetectable-Antivirus Steganography for Multimedia Information in PE-File. In: *2009 International Association of Computer Science and Information Technology - Spring Conference* [online]. [Singapur]: IEEE, Júl 2009, s. 437–441 [cit. 2022-05-05]. ISBN 978-0-7695-3653-8. Dostupné z: https://ieeexplore.ieee.org/document/5169389.

[7] Artz, D. Digital Steganography: Hiding Data Within Data. *IEEE Internet Computing* [online]. IEEE. Jún 2001, zv. 5, č. 3, s. 75–80, [cit. 2022-01-06]. DOI: 10.1109/4236.935180. ISSN 1941-0131. Dostupné z: https://ieeexplore.ieee.org/document/935180.

[8] Bhattacharyya, S., Banerjee, I. a Sanyal, G. A Survey of Steganography and Steganalysis Technique in Image, Text, Audio and Video as Cover Carrier. *Journal of*

Global Research in Computer Science [online]. Citeseer. Apríl 2011, zv. 2, č. 4, s. 1–16, [cit. 2022-01-28]. ISSN 2229-371X. Dostupné z: https://www.rroij.com/open-access/a-survey-of-steganography-and-steganalysis-technique-in-image-text-audio-and-video-as-cover-carrier-1-16.php?aid=37026.

[9] Bilal, M., Imtiaz, S., Abdul, W. et al. Chaos based Zero-steganography algorithm. *Multimedia Tools and Applications* [online]. Springer. September 2014, zv. 72, č. 2, s. 1073–1092, [cit. 2022-01-27]. DOI: 10.1007/s11042-013-1415-y. ISSN 1573-7721. Dostupné z: https://link.springer.com/article/10.1007/s11042-013-1415-y.

[10] Boelen, M. The 101 of ELF files on Linux: Understanding and Analysis. *Linux Audit: The Linux security blog about Auditing, Hardening, and Compliance* [online]. Máj 2019 [cit. 2022-03-05]. Dostupné z: https://linux-audit.com/elf-binaries-on-linux-understanding-and-analysis/. Path: Home; The 101 of ELF files on Linux: Understanding and Analysis.

[11] Corporation, I. *Intel 64 and IA-32 Architectures* [online]. Software Developer's Manual. Intel Corporation, september 2016 [cit. 2022-05-05]. Dostupné z: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-2b-manual.pdf#page=165.

[12] Demidenko, M., Kaipiyev, A., Naseri, M. V. et al. Understanding of PE Headers and Analyzing PE File. *PalArch's Journal of Archaeology of Egypt / Egyptology* [online]. November 2020, zv. 17, č. 7, s. 8611–8620, [cit. 2022-03-07]. ISSN 1567-214x. Dostupné z: https://www.archives.palarch.nl/index.php/jae/article/view/3670.

[13] Djebbar, F., Ayad, B., Hamam, H. et al. A view on latest audio steganography techniques. In: *2011 International Conference on Innovations in Information Technology* [online]. Abu Dhabi, United Arab Emirates: IEEE, Apríl 2011, s. 409–414 [cit. 2022-01-29]. ISBN 978-1-4577-0314-0. Dostupné z: https://ieeexplore.ieee.org/abstract/document/5893859.

[14] Djebbar, F., Ayad, B., Meraim, K. A. et al. Comparative study of digital audio steganography techniques. *EURASIP Journal on Audio, Speech, and Music Processing* [online]. Springer. Október 2012, zv. 25, č. 1, s. 1–16, [cit. 2022-01-29]. DOI: 10.1186/1687-4722-2012-25. ISSN 1687-4722. Dostupné z: https://asmp-eurasipjournals.springeropen.com/articles/10.1186/1687-4722-2012-25/.

[15] Douglas, M., Bailey, K., Leeney, M. et al. An overview of steganography techniques applied to the protection of biometric data. *Multimedia Tools and Applications* [online]. CrossMark. Júl 2018, zv. 77, č. 13, s. 17333–17373, [cit. 2022-01-26]. DOI: 10.1007/s11042-017-5308-3. ISSN 1573-7721. Dostupné z: https://link.springer.com/article/10.1007%2Fs11042-017-5308-3.

[16] El Khalil, R. a Keromytis, A. D. Hydan: Hiding Information in Program Binaries. In: Lopez, J., Qing, S. a Okamoto, E., ed. *Information and Communications Security* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, Október 2004, sv. 3269, s. 187–199 [cit. 2022-03-11]. Lecture Notes in Computer Science. ISBN 978-3-540-30191-2. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-540-30191-2_15.

[17] Evsutin, O., Melman, A. a Meshcheryakov, R. Digital Steganography and Watermarking for Digital Images: A Review of Current Research Directions. *IEEE Access* [online]. IEEE. September 2020, zv. 8, s. 166589–166611, [cit. 2022-01-06]. DOI: 10.1109/ACCESS.2020.3022779. ISSN 2169-3536. Dostupné z: https://ieeexplore.ieee.org/document/9187785.

[18] Febryan, A., Purboyo, T. W. a Saputra, R. E. Steganography Methods on Text, Audio, Image and Video: A Survey. *International Journal of Applied Engineering Research* [online]. Research India Publications. Január 2017, zv. 12, č. 21, s. 10485–10490, [cit. 2022-01-28]. ISSN 0973-4562. Dostupné z: https://www.semanticscholar.org/paper/Steganography-Methods-on-Text-%2C-Audio-%2C-Image-and-%3A-Febryan-Purboyo/c74a460cbf3263b3081bd56810e4968bccea12a5.

[19] Gori, G. Tattooing as a vehicle for secret messages in ancient Greece. *Tattoo Life* [online]. November 2021 [cit. 2022-01-22]. Dostupné z: https://www.tattoolife.com/tattooing-as-a-vehicle-for-secret-messages-in-ancient-greece/.

[20] Hamid, N., Yahya, A., Ahmad, R. B. et al. Image Steganography Techniques: An Overview. *International Journal of Computer Science and Security (IJCSS)* [online]. Citeseer. 2012, zv. 6, č. 3, s. 168–187, [cit. 2022-01-28]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.735.5356&rep=rep1&type=pdf.

[21] Hopper, N., Ahn, L. von a Langford, J. Provably Secure Steganography. *IEEE Transactions on Computers* [online]. IEEE. Máj 2009, zv. 58, č. 5, s. 662–676, [cit. 2022-01-24]. DOI: 10.1109/TC.2008.199. ISSN 1557-9956. Dostupné z: https://ieeexplore-ieee-org.ezproxy.lib.vutbr.cz/document/4663056.

[22] Hussain, M. a Hussain, M. A survey of image steganography techniques. *International Journal of Advanced Science and Technology* [online]. Islamabad, Pakistan: Citeseer. Máj 2013, zv. 54, s. 113–124, [cit. 2022-01-29]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.364.3275.

[23] Johnson, N. F. a Jajodia, S. Exploring steganography: Seeing the unseen. *Computer* [online]. IEEE. Február 1998, zv. 31, č. 2, s. 26–34, [cit. 2022-01-22]. DOI: 10.1109/MC.1998.4655281. ISSN 1558-0814. Dostupné z: https://ieeexplore.ieee.org/abstract/document/4655281.

[24] Johri, P., Mishra, A., Das, S. et al. Survey on Steganography Methods (Text, Image, Audio, Video, Protocol and Network Steganography). In: *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)* [online]. New Delhi, India: IEEE, Október 2016, s. 2906–2909 [cit. 2022-01-28]. ISBN 978-9-3805-4421-2. Dostupné z: https://ieeexplore.ieee.org/document/7724795.

[25] Kankowski, P. Redundancy of x86 Machine Code. *Strchr.com* [online]. 2009 [cit. 2022-05-05]. Dostupné z: https://www.strchr.com/machine_code_redundancy. Path: Home; Redundancy of x86 Machine Code.

[26] Ke, Y., Liu, J., Zhang, M.-Q. et al. Steganography Security: Principle and Practice. *IEEE Access* [online]. IEEE. November 2018, zv. 6, s. 73009–73022, [cit. 2022-01-25]. DOI: 10.1109/ACCESS.2018.2881680. ISSN 2169-3536. Dostupné z: https://ieeexplore.ieee.org/document/8537887.

[27] Kour, J. a Verma, D. Steganography Techniques – A Review Paper. *International Journal of Emerging Research in Management and Technology* [online]. Academia. Máj 2014, zv. 3, č. 5, s. 132–135, [cit. 2022-01-26]. ISSN 2278-9359. Dostupné z: https://www.academia.edu/40997885/Steganography_Techniques_A_Review_Paper?bulkDownload=thisPaper-topRelated-sameAuthor-citingThis-citedByThis-secondOrderCitations&from=cover_page.

[28] Kowalczyk, K. Portable Executable File Format. *Kowalczyk's Blog* [online]. Júl 2018 [cit. 2022-03-07]. Dostupné z: https://blog.kowalczyk.info/articles/pefileformat.html. Path: Home; File formats; pe format, pefile; Portable Executable File Format.

[29] Krishnan, R. B., Thandra, P. K. a Baba, M. S. An Overview of Text Steganography. In: *2017 Fourth International Conference on Signal Processing, Communication and Networking (ICSCN)* [online]. Chennai, India: IEEE, Marec 2017, s. 1–6 [cit. 2022-01-29]. ISBN 978-1-5090-4740-6. Dostupné z: https://ieeexplore.ieee.org/abstract/document/8085643.

[30] Marella, K. S. Steganography — 'The Dark cousin' of cryptography. *Techiepedia: WritingTech* [online]. Január 2021 [cit. 2022-01-22]. Dostupné z: https://medium.com/techiepedia/steganography-the-dark-cousin-of-cryptography-21d96a594068. Path: Home; CYBERSECURITY; Steganography — 'The Dark cousin' of cryptography.

[31] Mihara, T. Misdirection steganography. *Soft Computing* [online]. Springer. November 2020, zv. 24, č. 21, s. 16005–16010, [cit. 2022-01-27]. DOI: 10.1007/s00500-020-05345-1. ISSN 1433-7479. Dostupné z: https://link.springer.com/article/10.1007/s00500-020-05345-1.

[32] Mstafa, R. J., Elleithy, K. M. a Abdelfattah, E. Video Steganography Techniques: Taxonomy, Challenges, and Future Directions. In: *2017 IEEE Long Island Systems, Applications and Technology Conference (LISAT)* [online]. Farmingdale, NY, USA: IEEE, Máj 2017, s. 1–6 [cit. 2022-01-29]. ISBN 978-1-5386-3887-3. Dostupné z: https://ieeexplore.ieee.org/abstract/document/8001965.

[33] Nechta, I., Ryabko, B. Y., Andrey, F. et al. Stealthy Steganographic Methods for Executable Files. In: *XII International Symposium on Problems of Redundancy in Information and Control Systems* [online]. Saint-Petersburg, Russia: [b.n.], Máj 2009, s. 191–195 [cit. 2022-05-05]. ISBN 978-5-8088-0447-0.

[34] Nisi, D., Graziano, M., Fratantonio, Y. et al. Lost in the Loader: The Many Faces of the Windows PE File Format. In: ACM, ed. *24th International Symposium on Research in Attacks, Intrusions and Defenses* [online]. New York, NY, USA: Association for Computing Machinery, Október 2021, s. 177–192 [cit. 2022-03-06]. ISBN 9781450390583. Dostupné z: https://dl.acm.org/doi/10.1145/3471621.3471848.

[35] Nosrati, M., Karimi, R. a Hariri, M. An introduction to steganography methods. *World Applied Programming* [online]. Citeseer. August 2011, zv. 1, č. 3, s. 191–195, [cit. 2022-01-28]. ISSN 2222-2510. Dostupné z: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.208.5195&rep=rep1&type=pdf.

[36] Rakhi, S. G. A Review on Steganography Methods. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering* [online]. Citeseer. Október 2013, zv. 2, č. 10, s. 4635–4638, [cit. 2022-01-28]. ISSN 2278–8875. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1038.2695&rep=rep1&type=pdf.

[37] Sadek, M. M., Khalifa, A. S. a Mostafa, M. G. M. Video steganography: a comprehensive review. *Multimedia Tools and Applications* [online]. Springer. September 2015, zv. 74, č. 17, s. 7063–7094, [cit. 2022-01-28]. DOI: 10.1007/s11042-014-1952-z. ISSN 1573-7721. Dostupné z: https://link.springer.com/article/10.1007/s11042-014-1952-z.

[38] Sharma, S., Gupta, A., Trivedi, M. C. et al. Analysis of Different Text Steganography Techniques: A Survey. In: *2016 Second International Conference on Computational Intelligence Communication Technology (CICT)* [online]. Ghaziabad, India: IEEE, Február 2016, s. 130–133 [cit. 2022-01-29]. ISBN 978-1-5090-0210-8. Dostupné z: https://ieeexplore.ieee.org/abstract/document/7546588.

[39] Shih, F. Y. *Digital Watermarking and Steganography: Fundamentals and Techniques* [online]. 2. vyd. Boca Raton: CRC Press, apríl 2017 [cit. 2022-01-08]. 292 s. ISBN 9781315121109. Dostupné z: https://www.taylorfrancis.com/books/mono/10.1201/9781315121109/digital-watermarking-steganography-fundamentals-techniques-frank-shih.

[40] Shirali Shahreza, M. H. a Shirali Shahreza, M. A New Synonym Text Steganography. In: *2008 International Conference on Intelligent Information Hiding and Multimedia Signal Processing* [online]. Harbin, China: IEEE, August 2008, s. 1524–1526 [cit. 2022-01-29]. ISBN 978-0-7695-3278-3. Dostupné z: https://ieeexplore.ieee.org/abstract/document/4604331.

[41] Shirali Shahreza, M. Text Steganography by Changing Words Spelling. In: *2008 10th International Conference on Advanced Communication Technology* [online]. Gangwon, Korea (South): IEEE, Február 2008, sv. 3, s. 1912–1913 [cit. 2022-01-29]. ISBN 978-89-5519-136-3. Dostupné z: https://ieeexplore.ieee.org/abstract/document/4494159.

[42] Shirali Shahreza, M. a Shirali Shahreza, M. H. Text Steganography in SMS. In: *2007 International Conference on Convergence Information Technology (ICCIT 2007)* [online]. Gwangju, Korea (South): IEEE, November 2007, s. 2260–2265 [cit. 2022-01-29]. ISBN 0-7695-3038-9. Dostupné z: https://ieeexplore.ieee.org/abstract/document/4420590.

[43] Taha, M. S., Rahim, M. S. M., Lafta, S. A. et al. Combination of Steganography and Cryptography: A short Survey. *IOP Conference Series: Materials Science and Engineering* [online]. IOP Publishing. Máj 2019, zv. 518, č. 5, s. 052003–052016, [cit. 2022-01-24]. DOI: 10.1088/1757-899x/518/5/052003. ISSN 1757-8981. Dostupné z: https://doi.org/10.1088/1757-899x/518/5/052003.

[44] Tayel, M., Gamal, A. a Shawky, H. A proposed implementation method of an audio steganography technique. In: *2016 18th International Conference on Advanced Communication Technology (ICACT)* [online]. PyeongChang, Korea (South): IEEE,

Február 2016, s. 180–184 [cit. 2022-01-29]. ISBN 978-8-9968-6506-3. Dostupné z: https://ieeexplore.ieee.org/abstract/document/7423320.

[45] Tool Interface Standard (TIS). *Executable and Linking Format (ELF)* [online]. Specification. Tool Interface Standard (TIS), máj 1995 [cit. 2022-03-05]. Dostupné z: https://refspecs.linuxfoundation.org/elf/elf.pdf.

[46] Wahab, O. F. A., Khalaf, A. A. M., Hussein, A. I. et al. Hiding Data Using Efficient Combination of RSA Cryptography, and Compression Steganography Techniques. *IEEE Access* [online]. IEEE. Február 2021, zv. 9, s. 31805–31815, [cit. 2022-01-25]. DOI: 10.1109/ACCESS.2021.3060317. ISSN 2169-3536. Dostupné z: https://ieeexplore.ieee.org/document/9356603.

[47] Westfeld, A. a Pfitzmann, A. Attacks on Steganographic Systems: Breaking the Steganographic Utilities EzStego, Jsteg, Steganos, and STools – and Some Lessons Learned. In: Pfitzmann, A., ed. *Information Hiding* [online]. Berlin, Heidelberg: Springer, 2000, sv. 1768, s. 61–76 [cit. 2022-01-19]. Lecture Notes in Computer Science. ISBN 978-3-540-46514-0. Dostupné z: https://link.springer.com/chapter/10.1007/10719724_5.

[48] Yahya, A., Hamid, N., Ahmad, R. B. et al. Steganography Techniques. In: Yahya, A., ed. *Steganography Techniques for Digital Images* [online]. 1. vyd. Palapye, Botswana: Springer, Cham, Switzerland, 2019, kap. 2, s. 9–42 [cit. 2022-01-28]. ISBN 978-3-319-78597-4. Dostupné z: https://link.springer.com/book/10.1007/978-3-319-78597-4.

[49] Zaidan, A. A., Zaidan, B. B. a Othman, F. New Technique of Hidden Data in PE-File with in Unused Area One. *International Journal of Computer and Electrical Engineering* [online]. December 2009, zv. 1, č. 5, s. 642–650, [cit. 2022-05-05]. DOI: 10.7763/IJCEE.2009.V1.100. Dostupné z: http://www.ijcee.org/papers/100-650.pdf.

Príloha A

Obsah priloženého pamäťového média

```
/
├── config/
│   └── eq-classes.json          - Konfiguračný súbor
├── doc/                         - Zdrojové súbory dokumentácie
├── LICENSE
├── README.md
├── requirements.txt             - Použité moduly
├── src/                         - Zdrojové súbory
│   ├── analyzer.py
│   ├── args_parser.py
│   ├── common.py
│   ├── disassembler.py
│   ├── embedder.py
│   ├── eq_classes_processor.py
│   ├── extractor.py
│   ├── main.py
│   ├── my_instruction.py
│   └── selector.py
└── tests/                       - Zložka pre testovací skript
    ├── data/                    - Vkladané/extrahované dáta
    ├── executables/             - Používané krycie programy
    └── test.py                  - Testovací skript
```

Príloha B

Špecifikácia metód jednotlivých digitálnych steganografií

Obsahom tejto prílohy sú štyri podkapitoly, ktoré popisujú základné skupiny metód jednotlivých steganografií. Navyše, pri každej skupine je uvedených pár základných steganografických techník reprezentujúcich tieto skupiny. Ide o steganografiu textovú, obrazovú, zvukovú a videosteganografiu, ktorým sa venujú podkapitoly B.1, B.2, B.3 a B.4 v tomto poradí.

B.1 Textová steganografia

Táto podkapitola je doplnením podkapitoly 2.2 z hlavného textu práce. Jej obsahom je klasifikácia metód textovej steganografie, ktorá je nasledujúca:

B.1.1 Metódy založené na formáte

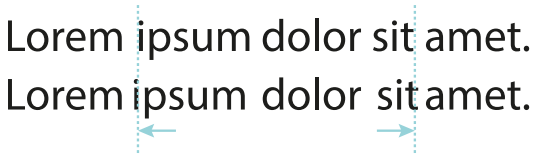
Tieto metódy používajú a menia formátovanie krycieho objektu (textu), čím skrývajú tajnú informáciu. Keďže ide len o formát textu, väčšina týchto metód nijako nemení samotný text krycieho objektu. Výnimkou môže byť napr. úmyselný preklep v rámci textu. Metódy tejto kategórie majú za cieľ byť spoľahlivo dekódovateľné, avšak pre čitateľa nerozoznateľné. Nasledujúce techniky sú len príkladmi tejto kategórie metód a je možné ich používať samostatne alebo aj spoločne: [8] [35] [24]

Metóda otvorených medzier (angl. *Open Space Method*)

Ide o pridávanie bielych znakov, konkrétne medzier, kde jedna medzera symbolizuje bit 0 a dve medzery bit 1. Tieto medzery sa môžu vyskytovať medzi slovami, vetami alebo odsekmi, alebo na konci riadkov. Aj keď je kapacita krycieho objektu výrazne obmedzená, metódu je možné použiť na akýkoľvek textový súbor, pričom odhalenie tajnej správy je veľmi obtiažne. Nevýhodou je, že niektoré textové editory môžu automaticky odstrániť nadbytočné medzery pri formátovaní, čím dôjde k strate ukrytých informácií. Taktiež veľkosť textového súboru sa vkladaním medzier zväčšuje. Napriek tomu má metóda využitie pri ukrytí informácií v rámci web stránky, pretože nadbytočné medzery v HTML dokumente neovplyvňujú zobrazenie výsledného dokumentu. [8] [38] [29]

Kódovanie s posunom slov (angl. *Word-Shift Coding*)

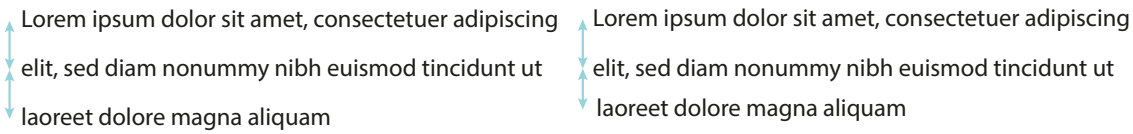
Úprava dokumentu horizontálnym posunutím slov v riadkoch zabezpečí jedinečné zakódovanie. Slová sa delia do skupín po troch v každom riadku, pričom krajné slová zostávajú bez narušenia. Stredné slovo sa posunie smerom doľava na zakódovanie bitu 0 alebo doprava na zakódovanie bitu 1. Riadky v rámci dokumentu sú zarovnané, čo zníži pravdepodobnosť odhalenia tejto metódy. Techniku je možné použiť na formátovaný súbor alebo bitmapu textového dokumentu, avšak použiteľná je len pre dokumenty s premenlivými medzerami medzi susednými slovami. Premenlivé medzery sú často používané kvôli rozdeleniu medzier pri zarovnaní textu. Pre extrakciu skrytej informácie je nutný pôvodný nezakódovaný dokument. Túto techniku znázorňuje obrázok B.1. [35] [29] [41]



Obr. B.1: **Ukážka kódovania s posunom slov** – vrchný riadok znázorňuje medzery medzi slovami pred zakódovaním informácie, pričom spodný po zakódovaní. (Prevzaté a upravené z [29])

Kódovanie s posunom riadkov (angl. *Line-Shift Coding*)

Jedinečné zakódovanie vytvorí úprava dokumentu vertikálnym posunutím riadkov textu. Ako pri predchádzajúcej metóde, aj tu existujú skupiny pozostávajúce z troch po sebe idúcich riadkov. Vždy krajné riadky zostávajú nenarušené, pričom stredný sa posunie smerom nahor, na zakódovanie bitu 0, alebo nadol, na zakódovanie bitu 1. Krajné riadky tak vytvárajú akési „čiary“, ktoré sa pri dekódovaní používajú na kontrolu, či bol stredný riadok posunutý alebo nie. Opäť je možné použiť bitmapu alebo formátovaný text. Existujú však prípady, kedy je možné dekódovanie úspešne uskutočniť bez prítomnosti pôvodného nezakódovaného dokumentu, pretože za bežných okolností je v celom dokumente rovnaké riadkovanie. Túto techniku znázorňuje obrázok B.2. [35] [29] [42]



Obr. B.2: **Ukážka kódovania s posunom riadkov** – obrázok vľavo znázorňuje riadkovanie pred zakódovaním informácie, pričom ten vpravo po zakódovaní. (Prevzaté a upravené z [29])

Kódovanie vlastností (angl. *Feature Coding*)

Opäť je možné kódovanie formátovaného súboru alebo bitmapy textového súboru. Cieľom je skúmanie vybraných textových prvkov, ktoré sa pozmenia alebo nepozmenia v závislosti od kódovanej informácie. Dekódovanie vyžaduje pôvodný dokument alebo presnejšiu špecifikáciu zmeny v pixeloch daného prvku. [35] [42]

Príkladom takejto metódy je *kódovanie zvislých koncových čiar písmen* (napr. b, d, h, k, . . .) ich predĺžením o jeden či viac pixelov, pre kódovanie bitu 0, a ich neporušením, pre kódovanie bitu 1. [29]

Iným použiteľným príkladom metódy, ktorá kóduje informáciu pomocou nejakej vlastnosti textu je tzv. *pohyb bodky v znakoch*. Malé písmená anglickej abecedy „i“ a „j“, rovnako ako arabské či perzské znaky abecedy, majú bodky. Bodky týchto znakov je možné posunúť nahor, pre zakódovanie bitu 0, alebo ich nechať nenarušené, pre zakódovanie bitu 1. Techniku znázorňuje obrázok B.3. [29]

Obr. B.3: **Ukážka kódovania s vybranou vlastnosťou textu** – písmeno vľavo má bodku neporušenú (bit 1), pričom písmeno vpravo ju má posunutú vyššie (bit 0). (Prevzaté a upravené z [29])

B.1.2 Náhodné a štatistické metódy

Tieto metódy generujú krycí text podľa štatistických vlastností daného jazyka a vkladajú tajnú informáciu tak, že sa vyskytuje v náhodnom poradí znakov. Toto poradie sa musí zdať náhodné aj pre prípadného útočníka. Metódy využívajú vzorové gramatiky určitého prirodzeného jazyka: [36] [24]

Pravdepodobnostná bezkontextová gramatika

Je to bežne používaný jazykový model, kde je priradená pravdepodobnosť pre každé transformačné pravidlo bezkontextovej gramatiky. Tento model je možné použiť na generovanie slovných sekvencií tak, že sa začne od koreňového uzla a rekurzívne sa aplikujú náhodne zvolené pravidlá. Vety sú zostavené podľa tajnej informácie, ktorá sa v nej má skrývať. Kvalita vygenerovaného stego-objektu priamo závisí od kvality použitých gramatík. [8]

Generovanie slov s rovnakými štatistickými vlastnosťami

Metóda generuje slová s rovnakými štatistickými vlastnosťami, ako je dĺžka slova či frekvencia písmen. Vygenerované slová často nemajú žiadnu lexikálnu hodnotu. [8]

B.1.3 Lingvistické metódy

Ide o umenie využívať prirodzený jazyk na ukrytie tajných informácií. Na úpravu textu sa využívajú jeho jazykové vlastnosti. Ide teda o kombináciu syntaktiky a sémantiky jazyka: [24] [36]

Syntaktická metóda

Pri tejto metóda sa umiestňujú interpunkčné znamienka (čiarka, bodka, . . .) na správne miesta tak, aby bolo možné ukryť tajnú informáciu. Metóda vyžaduje správnu identifikáciu

miest na umiestnenie týchto znamienok. Je však možné takto ukryť len malé množstvo informácií. [40] [41]

Sémantická metóda

Táto metóda skrýva informácie zámenou určitých slov za ich synonymá. Takáto substitúcia môže ukryť jeden alebo viac bitov tajnej informácie. Je tiež odolná voči prepisovaniu, čím do istej miery chráni tieto informácie. Niekedy však dôjde k zmene významu samotného textu. [38] [41] [40]

B.2 Obrazová steganografia

Doplnením podkapitoly 2.3 z hlavného textu práce sa zaoberá táto podkapitola. Jej obsahom je klasifikácia metód obrazovej steganografie, ktorá je nasledujúca:

B.2.1 Metódy priestorovej domény

Metódy priestorovej domény, nazývané aj *substitučné techniky*, sú skupinou relatívne jednoduchých techník, ktoré využívajú slabé stránky ľudského zraku. Preto je možné skryť informáciu do *najmenej významných bitov* (ďalej len LSB[1]) krycieho obrázka, čo predstavuje rovnomennú techniku. V rámci obrázka je možné brať LSB za náhodný šum, ktorý v ňom žiadne zmeny nepredstavuje. [20] [48] [22]

Algoritmus LSB môže mať pri vkladaní informácie do obrázka dve schémy:

- **Sekvenčná** – každý LSB obrázka je postupne nahradený bitmi informácie.
- **Rozptýlená** – bity informácie sú náhodne rozptýlené do LSB obrázka pomocou náhodnej sekvencie, ktorá toto vkladanie reguluje.

Steganografické nástroje založené na metódach LSB sú rôzne. Niektoré modifikujú LSB každého pixelu, pričom iné len vo vybraných oblastiach obrázka. Výhodou techník LSB je vysoká kapacita a nízka degradácia krycieho obrázka. Veľkou nevýhodou je, že nie sú robustné voči stratovej kompresii, ktorá je – ako už bolo zmienené – pri digitálnych obrázkoch veľmi častá. [20] [48] [22]

Prehľad niektorých použiteľných variantov LSB techník (okrem samotnej LSB metódy) [22] [36]:

- *Rozdiel hodnôt pixelov* (angl. *Pixel Value Differencing – PVD*)
- *Vkladanie dát založené na hranách* (angl. *Edges Based Data Embedding*)
- *Vkladanie do náhodných pixelov* (angl. *Random Pixel Embedding*)
- *Metóda založená na textúre* (angl. *Texture Based Method*)

B.2.2 Metódy transformačnej domény

V súčasnosti takmer všetky robustné steganografické algoritmy fungujú na vkladaní informácií v rámci *transformačnej domény*. Je to z dôvodu, že vkladanie dát do frekvenčnej oblasti signálu je dostatočne odolné v porovnaní s metódami časovej domény. Preto sa

[1] Najmenej významný bit (angl. *Least Significant Bit – LSB*)

občas v literatúre v súvislosti s touto skupinou metód objavuje termín *metódy frekvenčnej domény*. Techniky transformačnej domény sú výhodnejšie v porovnaní s technikami LSB, lebo skrývanie dát je zamerané na oblasti obrázka, ktoré sú menej vystavené kompresii či inému spracovaniu. Niektoré metódy tejto kategórie sú tiež nezávislé od formátu obrázka, čo implikuje odolnosť voči bezstratovej aj stratovej kompresii obrázka. Techniky transformačnej domény sa vo všeobecnosti delia na: [22] [36]

1. *Diskrétna Fourierová transformácia* (angl. *Discrete Fourier Transform – DFT*)
2. *Diskrétna kosínusová transformácia* (angl. *Discrete Cosine Transform – DCT*)
3. *Diskrétna vlnová transformácia* (angl. *Discrete Wavelet Transform – DWT*)

Najbežnejším obrazovým formátom používaným na internete je formát JPEG. Vo väčšine JPEG steganografických systémov sa informácie vkladajú do nenulových koeficientov DCT. Nasledujúce známe JPEG steganografické techniky sú toho dôkazom. [20] [48]

F5

Algoritmus vkladá informácie do absolútnej hodnoty nenulových koeficientov DCT znížením ich hodnoty o jedna namiesto náhrady LSB koeficientov DCT za bity informácie. Táto technika je absolútne imúnna voči vizuálnym útokom na stego-obrázok. Za účelom zníženia šumu zabudovaného do signálu sa používa maticové kódovanie. Ide o jednu z najpopulárnejších schém vkladania v doméne DCT. [20] [36]

OutGuess

Existujú dve verzie tejto techniky. Prvá, *OutGuess-0.13b*, je primárna verzia zobrazujúca štatistickú analýzu. Druhá, *OutGuess-0.2*, umožňuje ochranu voči štatistickým útokom. Implicitne sa technikou OutGuess myslí práve druhá verzia. Proces vkladania OutGuess pozostáva z náhodného vloženie bitov informácie do LSB koeficientov DCT, avšak s vyhýbaním sa nulám a jednotkám. Týmto vzniká komplexný histogram DCT pre stego-obrázok, ktorý je ekvivalentný s histogramom DCT pre ten pôvodný. Technika však imúnna voči vizuálnym útokom nie je. [20] [48]

MB

Technika založená na modeli *MB* môže byť definovaná ako všeobecný rámec na vykonávanie steganografie aj steganalýzy jednoduchým použitím štatistického modelu krycieho obrázku. Schéma MB má v prípade JPEG obrázkov vysokú kapacitu a je bezpečná voči štatistickým útokom prvého stupňa. [20] [48]

YASS

Alternatívny prístup k vkladaniu do JPEG obrázkov má schéma YASS[2]. Vstupný obrázok sa rozdelí do blokov s bezpečne veľkou veľkosťou – *B-bloky*. V ďalšom kroku sa náhodne vyberie v každom B-bloku podblok o rozmere 8×8 – *H-blok*. Pomocou kódov na opravu chýb sa šifrovaná informácia vloží do koeficientov DCT v H-bloku. Nakoniec sa obrázok komprimuje a je distribuovaný vo formáte JPEG po inverzii koeficientov DCT na H-blokoch. [20] [48]

[2]Ďalšia steganografická schéma (angl. *Yet Another Steganographic Scheme – YASS*)

Technika vlnovej transformácie

DWT konvertuje informácie o priestorovej doméne na informácie o frekvenčnej doméne. Vlny sa používajú v obraze, pretože DWT jasne rozdeľuje vysokofrekvenčné a nízkofrekvenčné informácie pixel po pixeli. Metóda DWT je uprednostňovaná pred metódou DCT vďaka rozlíšeniu, ktoré DWT poskytuje obrazu na rôznych úrovniach. Vlny sú matematické funkcie, ktoré rozdeľujú údaje na frekvenčné zložky, vďaka čomu sú ideálne na kompresiu obrazu. [36] [20] [48]

B.2.3 Metódy rozprestretého spektra

Schémy *metód rozprestretého spektra* spĺňajú maximálne požiadavky schém na skrytie informácií, najmä pokiaľ ide o štatistické hrozby. Z tohto dôvodu sú skryté údaje rozptýlené po celom obrázku bez zmeny štatistických vlastností. Celkovo možno metódy rozprestretého spektra použiť vo väčšine steganografických aplikácií, napriek tomu, že sú charakteristické tým, že sú vysoko matematickým a zložitým prístupom. Tieto metódy sa v steganografii opierajú buď o krycí obrázok ako šum, alebo sa pokúšajú pridať ku kryciemu obrázku pseudošum. [48] [20]

Krycí obrázok ako šum

Takýto steganografický systém zaobchádza s krycím obrázkom ako so šumom a môže tomuto obrázku priradiť jeden bit. Pre prípad prenosu viac bitov sa krycí obrázok rozdelí do krycích podobrázkov. Potom ide o steganografiu s rozprestretým spektrom priamej sekvencie. Keď sa krycie podobrázky skladajú zo samostatných bodov rozmiestnených po krycom obrázku, táto technika sa označuje ako steganografia s rozprestretým spektrom s preskakovaním frekvencie. Obe techniky sú odolné voči jemnej kompresii JPEG. [20]

Pseudošum

Technika ukazuje, že skrytá informácia je rozptýlená po celom krycom obrázku, a preto je ju ťažké odhaliť. Príkladom tejto techniky je *steganografia obrazu s rozprestretým spektrom*. Jej proces začína ukrytím informácie v šume a následne sa skombinuje s krycím obrázkom, čím sa dostane do stego-obrázku. Keďže sila vloženého signálu je oveľa nižšia ako sila krycieho obrázka, stego-obrázok sa stáva nepostrehnuteľným nielen pre ľudský zrak, ale aj pre steganalýzu bez prístupu k pôvodnému obrázku. [20]

V rámci obrazovej steganografie sa zistilo, že vysoké frekvencie zvyčajne pomáhajú pri neviditeľnosti skrytých informácií, no zároveň nie sú veľmi robustné. Naopak, nízke frekvencie podmieňujú lepšiu robustnosť na úkor viditeľnosti, čo znamená, že sú nepoužiteľné. Túto konfliktnú situáciu zosúlaďuje technika rozprestretého spektra tým, že umožňuje vložiť nízkoenergetický signál do každého z frekvenčných pásiem. Techniky rozprestretého spektra je tiež možné kombinovať s transformačnými technikami, aby sa zvýšila kapacita. [20]

B.2.4 Štatistické metódy

Ide o techniky, ktoré upravujú štatistické vlastnosti obrázka v procese vkladania. Štatistické steganografické techniky označujú použitie existencie jednobitového steganografického systému, v ktorom je jeden bit dát vložený do obrázka. Proces vloženia pozostáva z jednoduchej a malej úpravy obrázka tak, aby nastala významná zmena v štatistických charakteristikách – vtedy ide o zakódovanie bitu 1. Ak obrázok zostane nezmenený, znamená to zakódovanie

bitu 0. Je tiež možné zakódovať viacbitovú informáciu, kedy sa obrázok rozdelí na samostatné bloky (podobrázky), pričom každý predstavuje jeden bit informácie. [20] [48]

Inou štatistickou metódou je použitie vodoznaku, ktorý poskytuje základ pre štatistickú funkciu. Keďže sa vodoznaky považujú za ťažko rozpoznateľné a ťažko odstrániteľné, pričom sa dajú ľahko obnoviť – za predpokladu znalosti kľúča. Bloky obrázka, ktoré majú zakódovať bit 1 sa označia vodoznakom. Bloky, ktoré nie sú označené vodoznakom, kódujú bit 0. Niekedy sa v literatúre táto metóda radí do skupiny *maskovacích a filtrovacích metód* (angl. *Masking and Filtering Methods*), ktorých výhodou oproti LSB technikám je, že maskujú vkladanú informáciu vo viditeľných častiach obrázku a nie na úrovni šumu. Tým sa stavajú odolnými voči stratovej kompresii (napr. JPEG) a rôznym spracovaniam obrázku. [48] [35]

Štatistické metódy sú náchylné na útoky orezania, rotácie a zmeny mierky obrázka, a tiež na útoky, na ktoré je náchylný samotný vodoznak. Účinnou obranou voči takýmto útokom môže byť koncept, ktorý vytvára bloky v obrázku na základe jeho obsahu (napr. bloky predstavujú tváre na obrázku) a používa kódovanie na opravu chýb v rámci vkladanej informácie. Tento koncept môže nadobudnúť robustnosť rovnakú, akú má vodoznak. Avšak tieto metódy nie sú odolné voči steganalýze, ktorá meria štatistické vlastnosti obrázku, preto sú štatistické metódy v praxi menej použiteľné v porovnaní s inými metódami. [48] [20]

B.2.5 Techniky skreslenia

Techniky skreslenia vyžadujú znalosť pôvodného krycieho obrázku kvôli procesu dekódovania, kde dekodér funguje na kontrole rozdielov medzi pôvodným obrázkom a skresleným obrázkom, čím sa dekóduje vložená informácia. Vloženie informácie teda znamená skreslenie obrázku. Skreslenie obrázku je vykonané na základe kódovanej informácie. Tá sa kóduje v náhodne vybraných pixeloch. Ak sa stego-obrázok líši od pôvodného v danom pixeli, znamená to, že tento pixel kóduje bit 1, inak kóduje bit 0. Výhodou techniky je možnosť upraviť pixeli tak, aby sa zachovali štatistické vlastnosti obrázka, čím sa technika stáva odolnou voči štatistickým útokom. Vďaka tomu sa techniky skreslenia líšia od metód LSB. [20] [22] [48]

Nevýhodou, ktorá komplikuje použitie tejto techniky, je nutnosť odoslať aj pôvodný obrázok. Taktiež by sa jeden krycí obrázok nemal použiť viackrát – platí pre každú steganografickú techniku. V niektorých prípadoch, keď je informácia vložená kódovaním na opravu chýb, je táto technika odolná voči útokom pozmeňujúcim stego-obrázok, pretože vloženú informáciu je možné úplne obnoviť. [20] [22] [48]

B.3 Zvuková steganografia

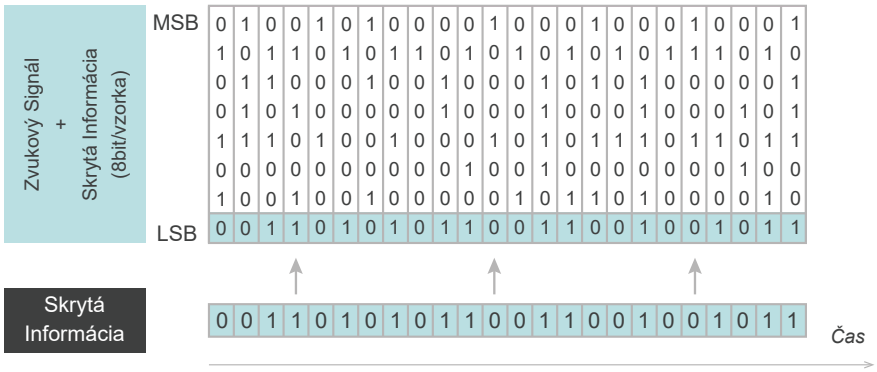
Podkapitoly 2.4 z hlavného textu práce dopĺňa táto podkapitola. Jej obsahom je klasifikácia metód zvukovej steganografie, ktorá je nasledujúca:

B.3.1 Metódy časovej domény

Väčšina metód tejto kategórie využíva techniky LSB a jej variantov. Pre túto skupinu metód nie sú hlavnými znakmi robustnosť ani bezpečnosť, no technika LSB a jej varianty poskytujú jednoduchý spôsob ukrytia informácií. V súčasnosti bolo vyvinutých len niekoľko metód tejto kategórie, pričom nasledujúce uvedené techniky sú len niektorými z nich: [14]

Kódovanie najmenej významných bitov (angl. *LSB Coding*)

Jednou z prvých, najjednoduchších a bežne používaných techník pre zvukovú steganografiu je práve kódovanie LSB. Technika pozostáva z vloženia každého bitu informácie do bitu LSB krycieho zvukového signálu. Aj keď je táto metóda veľmi jednoduchá, nedokáže ochrániť skrytú informáciu ani pred malými úpravami, ktoré môžu vzniknúť za rôznych situácií (napr. konverzia formátu, ...). Technika LSB sa dá ľahko implementovať a kombinovať s inými efektívnejšími technikami ukrývania informácií. Jej ďalšou výhodou je vysoká kapacita na prenos mnohých typov digitálnych objektov, avšak dĺžka tajnej informácie by mala byť menšia než celkový počet vzoriek krycieho signálu. LSB taktiež využíva nedokonalosť ľudského sluchu, ktorý nevie rozpoznať malé odchýlky frekvencií zvukového signálu. Okrem toho je LSB veľmi rýchla a efektívna technika, pričom sa kvalita signálu neznižuje. Techniku LSB znázorňuje obrázok B.4. [44] [8]



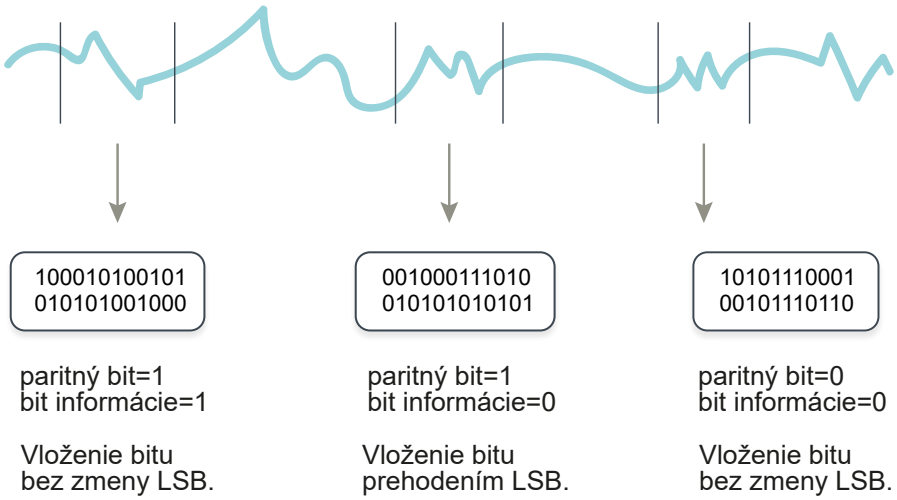
Obr. B.4: **Ukážka kódovania LSB** (prevzaté a upravené z [14])

Skrytie ozveny (angl. *Echo Hiding*)

Tajné informácie sú vložené ako ozvena do krycieho zvykového signálu. Ozvena je rezonancia pridaná ku kryciemu signálu, čo spôsobí, že sa predíde problémom s aditívnym šumom. Na úspešné skrytie informácie je potrebné zmeniť počiatočnú amplitúdu, rýchlosť doznievania (poklesu) a uskutočniť posun (oneskorenie) od pôvodného signálu tak, aby sa ozvena stala počuteľnou a tým reprezentuje zakódovanú tajnú informáciu. Ozvenu nie je možné ľahko detegovať, pretože všetky tri parametre sú pod hranicou ľudského sluchu. Kvôli nízkej bezpečnosti a rýchlosti sa vo výskume týchto techník ďalej nepokračuje. [44] [35]

Paritné kódovanie (angl. *Parity Coding*)

Tieto techniky pracujú so skupinami vzoriek namiesto jednotlivých. Teda jednotlivé vzorky sa zoskupia a vypočíta sa ich parita. Na vkladanie bitov informácie po jednom sa kontroluje paritný bit skupiny vzoriek. Ak sa tento bit zhoduje s bitom informácie, nedeje sa nič. Ak sa však tieto bity nezhodujú, zmení sa LSB ktorejkoľvek z jednotlivých vzoriek v rámci danej skupiny tak, aby sa paritný bit rovnal bitu informácie. Túto techniku znázorňuje obrázok B.5. [44] [18]



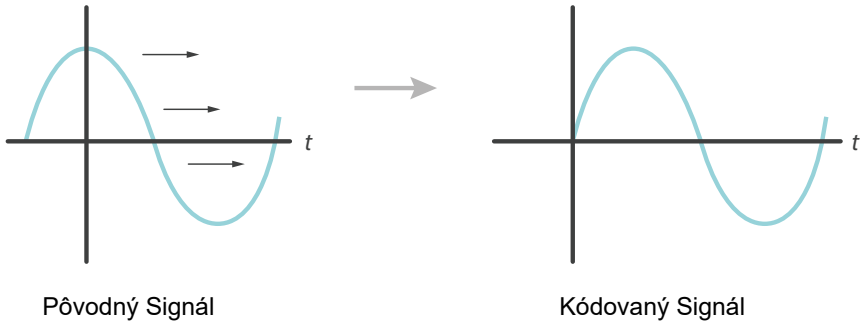
Obr. B.5: **Ukážka paritného kódovania** (prevzaté a upravené z [44])

B.3.2 Metódy transformačnej domény

Táto skupina metód využíva tzv. „fenomén maskovacieho efektu“, kedy sa maskujú slabšie frekvencie blízko silnejších rezonančných. Tento fenomén využíva nedokonalosti ľudského sluchu a umožňuje tak veľmi efektívne skrytie informácií. Je známe, že skrytie informácie v transformačnej doméne, na rozdiel od časovej, poskytne lepšie výsledky z hľadiska pomeru signálu k šumu. V poslednom čase bolo vyvinutých veľa techník tejto kategórie, ktorým sa podarilo lepšie realizovať bezpečnosť a robustnosť. Preto sú do určitej miery skryté informácie odolné voči prevzorkovaniu, filtrácii alebo zosilneniu zvukového signálu. Na druhú stranu, pravdepodobne nepreživijú hlučné prenosové prostredie alebo kompresiu. Následne sú uvedené len niektoré techniky z tejto kategórie metód: [14]

Fázové kódovanie (angl. *Phase Coding*)

Ľudský sluch nevie rozpoznať fázovú zmenu zvukového signálu tak, ako dokáže rozpoznať šum v signáli. Preto táto metóda túto skutočnosť využíva. Technika zakóduje bity tajnej informácie ako fázové posuny vo fázovom spektre digitálneho signálu. Tým sa dosiahne nepočuteľné kódovanie v zmysle pomeru signálu k vnímanému šumu, preto vzniká odolnosť voči steganalýze založenej na šume. Fázové kódovanie týmto rieši nevýhody metód zvukovej steganografie vyvolávajúcich šum. Technika je odolná voči skresleniu signálu, ale nevydrží dolnopriepustnú filtráciu. Techniku znázorňuje obrázok B.6. [8] [35] [44]



Obr. B.6: **Ukážka posunu fázy zvukového signálu vpravo** (prevzaté a upravené z [44])

Kódovanie rozprestretého spektra (angl. *Spread Spectrum Coding*)

Pri základnej metóde sa náhodne rozložia bity tajnej informácie cez frekvenčné spektrum zvukového signálu. Táto metóda, na rozdiel od kódovania LSB, šíri tajnú informáciu pomocou kódu. Tento kód je nezávislý od skutočného krycieho signálu a je známy odosielateľovi aj príjemcovi. Táto metóda môže fungovať lepšie ako techniky kódovania LSB a fázy vďaka miernej rýchlosti prenosu dát v spojení s vysokou úrovňou odolnosti voči technikám steganalýzy. Avšak, podobne ako metóda kódovania LSB, aj táto metóda môže do zvukového signálu vniesť šum. Vytvára sa tak zraniteľnosť, ktorú je možné využívať pri steganalýze. [8] [44]

Techniky vlnovej domény (angl. *Wavelet Domain Methods*)

Ide o zvukovú steganografiu založenú na diskrétnej vlnovej transformácii. Informácie sa skrývajú v LSB koeficientov vlnovej transformácie audio signálu, pričom sa dosahuje vysokej kapacity až 200 kbps v 44, 1 kHz zvukovom signáli. Pre zlepšenie nepostrehnuteľnosti vloženej informácie je možné použiť prah počutia pri vkladaní informácií do celočíselných koeficientov. Skrytie informácií vo vlnovej doméne je síce rýchle, no extrakcia informácie príjemcom nemusí byť presná. [14] [13]

B.4 Videosteganografia

Posledná podkapitola, ktorá dopĺňa podkapitolu 2.5 z hlavného textu práce. Jej obsahom je klasifikácia metód videosteganografie, ktorá je nasledujúca:

B.4.1 Substitučné metódy

Techniky tejto kategórie metód sú založené na substitúcii a nahrádzajú redundantné informácie krycieho objektu za požadovanú tajnú správu. Ich výhodami sú vysoká kapacita pre vložené informácie a jednoduchosť implementácie v porovnaní s ostatnými. Nasledujúce techniky sú len príkladmi substitučných metód. [37]

Technika najmenej významného bitu (angl. *LSB Technique*)

Technika dokáže skryť pomerne veľké množstvo bitov tajnej informácie nahradením niektorých najmenej významných bitov jednotlivých pixelov krycieho videa. Experimenty v [37] dokazujú, že maximálny počet nahradených LSB bitov sú štyri. Je to z dôvodu začínajúceho

vizuálneho skreslenia krycieho videa, ktoré je samozrejme závislé od použitých farieb vkladanej informácie (napr. obrazovej). Technikou LSB je inšpirovaná väčšina substitučných metód. Existuje niekoľko variantov metódy LSB od takých, ktorých vizuálne skreslenie optimalizované je, no nie sú dostatočne robustné alebo majú nízku kapacitu, až po také, ktoré značne modifikujú niektoré vlastnosti základnej techniky LSB. [37]

Napokon jednoduchá implementácia a nízka výpočtová náročnosť techniky LSB pritiahla pozornosť a začala sa využívať na steganografiu v reálnom čase. Ide o ukrytie tajnej informácie v snímkach reklamných bilbordov, kde každá snímka je rozdelená na malé bloky, v ktorých je ukrytá informácia. Na zabezpečenie je možné použiť tajný kľúč. Takúto techniku je možné použiť na vysielanie tajnej informácie na verejných miestach (parky, obchodné centrá, ...). Technika nepotrebuje žiaden úložný priestor, čo je veľmi výhodné pri skrývaní informácie v reálnom čase. Keďže nie je k dispozícii ani krycí objekt, nie je možné vykonávať jeho analýzu. Jediným obmedzením je, pochopiteľne, nutnosť pripojiť elektronické zariadenie implementujúce takúto techniku k zariadeniu vysielajúcemu v reálnom čase. Pre extrakciu informácie si príjemca musí zaznamenávať snímky obrazu. Ak zmešká nejakú snímku obsahujúcu informáciu, prichádza o ňu. [37]

Vo všeobecnosti je myšlienkou metódy LSB nahradenie najmenej významných bitov za iné, čo vedie k zhoršeniu kvality krycieho videa. Tento problém sa snažia riešiť nasledujúce metódy. [37]

Segmentácia zložitosti bitovej roviny (angl. *Bit Plane Complexity Segmentation – BPCS*)

Táto metóda využíva slabé stránky ľudského zraku, ktorý nedokáže prijímať informácie o obraze v komplikovanom binárnom vzore. BPCS dokáže pracovať v priestorovej aj v transformačnej doméne. Myšlienkou metódy BPCS je rozdeliť snímku do bitových rovín. Bitová rovina sa môže považovať za výrez snímku, ktorý je tvorený všetkými bitmi špecifickej významovej pozície z každej binárnej číslice. Keď sú identifikované bitové roviny snímky, zmeria sa zložitosť každej oblasti tejto roviny. Oblasti sú rozdelené do dvoch typov: [37]

1. *informatívna oblasť* a
2. *oblasť podobná šumu*

Informatívna oblasť zostáva neporušená, no do oblasti pripomínajúcej šum sa vkladajú bity tajnej informácie, čo ma za následok minimálne zníženie kvality videosnímku.

Trojcestný rozdiel hodnoty pixelov (angl. *Tri-way Pixel-Value Differencing – TPVD*)

Ide o modifikáciu známej metódy *rozdiel hodnoty pixelov* (angl. *Pixel-Value Differencing – PVD*). Ukrytie tajnej informácie pomocou PVD je založené na rozdiele hodnôt dvoch susedných pixelov. Hodnoty týchto rozdielov sa delia do rozsahov, pričom sa každý skladá z dolnej hranice, hornej hranice a šírky rozsahu. Menší index rozsahu indikuje hladkú oblasť a vyšší ostrú oblasť. Väčšie množstvo informácie je možné vložiť do ostrej oblasti na rozdiel od tej hladkej. [37]

Pred ukrytím údajov sa krycia snímka rozdelí na neprekrývajúce sa bloky dvoch susedných pixelov. Následne sa určí hodnota rozdielu a rozsah. Potom sa vypočíta počet bitov informácie, ktoré sa môžu skryť, a to na základe indexu rozsahu. V tejto chvíli sa zistený

počet bitov z informácie extrahuje a ukryje. Ich príslušná desatinná hodnota sa ďalej použije na vytvorenie nového rozdielu, podľa ktorého sa upravia hodnoty pixelov. Technika TPVD vkladá informáciu do všetkých vertikálnych, horizontálnych a diagonálnych okrajov, čo zvýši kapacitu krycieho snímku. [37]

B.4.2 Metódy transformačnej domény

Napriek rôznym modifikáciám, substitučné algoritmy neustále bojujú so slabou odolnosťou voči modifikácii krycieho objektu, ako sú kompresia, zmena formátu atď. *Metódy transformačnej domény* sú síce zložitejšie, no s vyššou robustnosťou a transparentnosťou vnímania kvality vzniknutých stego-objektov. Zásadou každej techniky tejto kategórie je transformácia krycieho videa do frekvenčnej domény s následným vložením tajnej informácie do niektorých alebo všetkých transformovaných koeficientov. Posledným krokom na záver je spätná transformácia zmenených koeficientov do pôvodného krycieho videa. Takéto transformácie je možné uskutočniť pomocou DFT, DCT a DWT. Vo videosteganografii sa častejšie používajú metódy založené na DCT a DWT. Metódy DFT sa ukázali ako neoptimálne, pretože zavádzajú veľké zaokrúhľovacie chyby. [37]

B.4.3 Adaptívne metódy

Adaptívne metódy sú novou technikou vkladania, hovorí sa im tiež *maskovacie metódy*. Jej myšlienkou je analýza štatistických znakov krycieho videosúboru pred vložením tajnej informácie. Výsledkom analýzy je identifikácia najvhodnejších oblastí pre vloženie informácie – *oblasti záujmu*. Okrem toho môže byť výstupom analýzy aj počet bitov informácie, ktoré sa majú skryť. Počet závisí od funkcie adaptívnej kapacity. V podstate je táto skupina metód len špeciálnym prípadom techník z ostatných kategórií tejto klasifikácie. V konečnom dôsledku pre dosiahnutie lepšej kvality stego-videa je krycie video adaptívne upravené podľa niekoľkých kritérií. Svoj adaptívny variant má aj metóda LSB. [37]

B.4.4 Metódy založené na formáte

Ako z názvu vyplýva, tieto techniky sú metódy navrhnuté priamo pre konkrétny formát videosúboru. V súčasnosti je navrhnutých viacero použiteľných formátov, ktoré je takto možné použiť ako krycie videá. Jedným z najnovších štandardov kompresie videosúboru je H.264/AVC, ktorý poskytuje vysokú účinnosť kompresie a je vhodný pre rýchly prenos po sieti. Metód založených na tomto formáte je viacero, pretože môžu efektívne využívať jeho štruktúru. [37]

Ďalším užitočným formátom je formát videosúborov Flash (prípona .FLV). Obľúbenosť formátu na internete je do značnej miery zapríčinená jeho jednoduchou štruktúrou a malou veľkosťou v porovnaní s inými. Príklad algoritmu pre tento formát je založený na myšlienke rovnomerného rozdelenia tajnej informácie medzi značky (angl. *tags*) videa v celom súbore, pričom jednotlivé časti informácie sú vložené za každú značku tak, aby sa skutočné značky videa a jeho zvuku neovplyvnili – nemodifikovali a ani nevynechali. Týmto zostáva kvalita videa úplne nezmenená a bez akéhokoľvek skreslenia. Výhodou je, že je možné vložiť informáciu neobmedzenej veľkosti, s čím sa ale zvyšuje veľkosť krycieho videa. Navyše, algoritmus nie je robustný. [37]

B.4.5 Metódy generujúce krycí videosúbor

Všetky vyššie uvedené metódy využívajú určitý krycí objekt, na ktorom aplikujú steganografický algoritmus. Táto sekcie približuje metódy, ktoré syntetizujú objekt na to, aby ho mohli použiť ako krycí v rámci výmeny tajných informácií. V tomto prípade ide o myšlienku generovania dynamického krycieho videa. Tento proces si vyžaduje použitie tajného steganografického kľúča a tajnej informácie. Generovanie krycieho videosúboru predstavuje funkciu

$$X(A, D)$$

kde X je funkcia, ktorá generuje krycí videosúbor na základe tajnej informácie. Parameter D predstavuje bity vkladanej tajnej informácie a parameter A predstavuje počet vzoriek potrebných pre ukrytie bitov D. [37] [18]

Tieto metódy vyžadujú na vstup databázu obrázkov, z ktorých sa zhromaždí požadovaný počet pre vygenerovanie krycieho videa. Výhodou takýchto metód je, že útočníkovi neposkytujú pôvodné obrázky. Naopak, nevýhodou je, že technika môže vyvolať podozrenie u útočníka, ak zhromaždená sekvencia obrázkov je voči sebe irelevantná. Alternatívou k tomu môže byť zhromaždenie obrázkov, ktoré budú tvoriť prezentáciu sprevádzanú zvukom. [37]