Univerzita Komenského
Fakulta Matematiky, Fyziky a Informatiky
Katedra Informatiky

Ivan Kohút

Watermarking spustiteľného kódu

Diplomová práca

Bratislava 2007

Watermarking spustiteľného kódu

Diplomová práca

Ivan Kohút

Univerzita Komenského v Bratislava
Fakulta Matematiky, Fyziky a Informatiky
Katedra informatiky

Odbor: Informatika

Diplomový vedúci: Mgr. Miroslav Demeter

Bratislava 2007

Čestne prehlasujem, že túto diplomovú prácu som vypracoval samostatne len s použitím uvedenej literatúry a s odbornou pomocou vedúceho práce.

Bratislava, máj 2007 Ivan Kohút

Poďakovanie

Ďakujem svojmu vedúcemu diplomovej práce, Mgr. Miroslavovi Demeterovi, a Radovanovi Mikušovi za mnoho cenných rád, podnetov a pripomienok pri písaní tejto práce.

Abstrakt

Watermarking spustiteľného kódu je jedným z nástrojov boja proti neoprávnenému kopírovaniu a používaniu softvéru. Zaoberá sa vkladaním tajných správ, vodoznakov, do programov. Vodoznakom je najčastejšie informácia o autorských právach vzťahujúcich sa k programu alebo informácia identifikujúca zákazníka. V prípade krádeže intelektuálneho vlastníctva je možné pomocou vodoznaku dokázať vlastníctvo softvéru alebo vystopovať osobu, ktorá poskytla softvér pre nelegálnu distribúciu.

V prvej časti tejto práce sa venujeme opisu existujúcich techník watermarkingu softvéru. Väčšina z nich má nedostatky, ktoré umožňujú vodoznak jednoduchším či zložitejším spôsobom znehodnotiť. V roku 2004 bola vyvinutá sľubná technika Dynamic Path-Based Software Watermarking (DPB) [4], voči ktorej však bol vymyslený a úspešne aplikovaný hybridný staticko-dynamický útok [9]. V druhej časti práce sa preto venujeme analýze uvedeného útoku a vychádzajúc z techniky DPB, návrhu novej techniky, ktorá nemá niektoré z jej nedostatkov. Preto je odolnejšia voči hybridnému staticko-dynamickému útoku a vďaka svojmu predchodcovi aj voči mnohým iným.

Kľúčové slová: Intelektuálne vlastníctvo, Ochrana softvéru, Watermarking

Abstract

Software watermarking is one of the means to fight against unauthorized software copying and usage. It deals with embedding secret messages, watermarks, into programs. Watermark is usually a copyright notice, which relates to program, or customer identification information. In case of intellectual property theft the watermark allows to prove ownership or to trace a person who provided program for illegal distribution.

In the first part of this thesis we describe existing software watermarking techniques. Most of them have shortcomings which allows for watermark destruction in more or less difficult way. In 2004 promising technique called Dynamic Path-Based Software Watermarking (DPB) [4] was developed against which Hybrid Static-Dynamic attack [9] was developed and succesfully apllied. In the second part of this thesis we perform an analysis of said attack and present a new technique based on DPB which lacks some of DPB's design weaknesses. It is more resilient against the Hybrid Static-Dynamic Attack and against many others thanks to its ancestor.

Keywords: Intellectual property, Software protection, Watermarking

Obsah

Úvod **9**

1 Základné definície a pojmy **11**
1.1 Kategórie watermarking techník . . . 12
1.2 Požiadavky kladené na watermarking techniky . . . 13
1.3 Kritéria kvality . . . 13
1.4 Útoky voči watermarking technikám . . . 15
1.4.1 Odstránenie vodoznaku . . . 15
1.4.2 Skreslenie programu . . . 15
1.4.3 Pridanie ďalšieho vodoznaku . . . 16
1.5 Ochrana proti útokom . . . 16

2 Existujúce techniky **18**
2.1 Statické techniky . . . 18
2.1.1 Statické dáta . . . 19
2.1.2 Digitálne média uložené v dátovej sekcií . . . 19
2.1.3 Usporiadanie základných blokov . . . 20
2.1.4 Alokácia registrov . . . 20
2.1.5 Frekvencie skupín inštrukcií . . . 20
2.1.6 Nezrozumiteľné predikáty . . . 21
2.1.7 Statická štruktúra behu programu . . . 21
2.2 Dynamické techniky . . . 21
2.2.1 Easter Egg . . . 22
2.2.2 Dynamické dáta . . . 22
2.2.3 Dynamic execution trace . . . 22
2.2.4 Topológia dynamicky vybudovaného grafu . . . 22
2.2.5 Watermarking použitím výrezov programov (SWuS) . . . 23

2.2.6 Dynamic Path-Based Watermarking (DPB) 26
2.3 Hybridné techniky 28
2.3.1 Abstraktný watermarking 28
2.4 Zhrnutie 29

3 Návrh watermarking techniky 31
3.1 Hybridný staticko-dynamický útok 31
3.1.1 Konštrukcia grafu toku riadenia 32
3.1.2 Možnosti útoku 33
3.1.3 Aplikácia útoku na DPB 33
3.1.4 Návrh novej techniky 34
3.2 Základný princíp novej techniky 35
3.3 Vloženie vodoznaku do programu 35
3.3.1 Rozdelenie vodoznaku 35
3.3.2 Kódovanie (úsekov) vodoznaku 37
3.3.3 Výber WM funkcií 39
3.3.4 WM funkcie 40
3.4 Rozpoznávanie vodoznaku 46
3.4.1 Konštrukcia grafu volania funkcií programu 46
3.4.2 Rozpoznanie WM funkcií 46
3.4.3 Konštrukcia vodoznaku 50
3.5 Odolnosť voči útokom 50
3.5.1 Odstránenie vodoznaku 50
3.5.2 Skreslenie vodoznaku 53
3.5.3 Pridanie ďalšieho vodoznaku 53
3.6 Ďalšie kritériá kvality 54
3.6.1 Kapacita 54
3.6.2 Efektivita 54
3.7 Porovnanie techník 55
3.7.1 Utajenosť 55
3.7.2 Odolnosť 55
3.7.3 Kapacita 56
3.7.4 Efektivita 56
3.7.5 Zhrnutie 57
3.8 Požiadavky 58
3.9 Príklad použitia 60

Záver **62**

Literatúra **63**

Zoznam obrázkov

2.1 Priradenie do premennej chránené podmienkou G_P (Guard Predicate) . . . 23
2.2 Rozšírenie o podmienku O_P s pravdivostnou hodnotou **true** 24
2.3 Nahradenie priradenia dvojicou priradení 24
2.4 Pomocné funkcie F_1 a F_2 25
2.5 Úprava podmienkového príkazu 25
2.6 Použitie vetviacej funkcie 27
2.7 Volanie vetviacej funkcie 28

3.1 Usporiadanie vrcholov prechodom grafu 38
3.2 Výpočet adresy volanej funkcie 41
3.3 Volanie WM funkcií v cykle 42
3.4 Volanie WM funkcií v rade 42
3.5 Výpočet adresy volanej WM funkcie 43
3.6 Funkcia WM-Double v. 1 44
3.7 Funkcia WM-Double v. 2 45
3.8 Graf volania funkcií 48

Úvod

Pojem vodoznak sa väčšine ľudí spája s ochranou bankoviek. Najčastejšie sú to rôzne pásiky, obrazce, ktoré na prvý (alebo n-tý) pohľad nie je vidieť. Ich podstatnou vlastnosťou je, že sa nedajú skopírovať, a teda neprítomnosť vodoznaku dokazuje, že skopírovaná bankovka je falošná, t. j. nie je vydaná centrálnou bankou.

V informatike sa vodoznaky používajú na ochranu elektronických diel. Tu však podoba s papierovými bankovkami končí, pretože pri kopírovaní digitálnych filmov, obrázkov, hudobných súborov či programov sa kvalita nemení, teda aj vodoznak sa skopíruje dokonale presne. Cieľom digitálnych vodoznakov preto nie je ochrana diela voči kopírovaniu, ale možnosť dokázať, kto je skutočným autorom daného diela. Kópia bankovky by mala byť, aj napriek snaženiu falšovateľa, vďaka nekvalitne okopírovaným vodoznakom, odlíšiteľná od originálu. Naproti tomu kópia digitálneho diela by mala vodoznak uchovať aj v prípade, že sa ho útočník snaží odstrániť. Preto, na rozdiel od vodoznakov v bankovkách, sa snažíme digitálne vodoznaky skryť, aby sme ich ochránili pred odstránením.

V súčasnosti sú najrozvinutejšou oblasťou digitálneho watermarkingu multimédiá, a to teoreticky (množstvo článkov, konferencie) aj prakticky (dostupný softvér a aj jeho používanie). Naproti tomu watermarking softvéru (Software watermarking), oblasť informatiky zaoberajúca sa vodoznakmi pre programy, je relatívne mladá a málo rozvinutá, počiatky siahajú do začiatku deväťdesiatych rokov. Je trochu prekvapujúce, ako málo pozornosti sa tejto oblasti venuje, vzhľadom na jej potenciál a rozšírenosť neoprávneného používania a obchodovania so softvérom.

Prvým cieľom tejto diplomovej práce je vytvoriť prehľad v oblasti watermarking-u softvéru. Je dôležitý preto, lebo táto oblasť je mladá a dosť zanedbávaná (v porovnaní s watermarkingom digitálnych médií). Venuje sa jej iba niekoľko vedcov a aj to často ako vedľajšej oblasti ich skúmania. A preto hoci výsledkov ich práce nie je veľa, človeku neznalému v tejto časti informatiky zaberie veľa času získať medzi nimi prehľad.

Tak ako každá diplomová práca, aj táto má ambíciou priniesť do skúmanej oblasti niečo nové. V tomto prípade sme sa sústredili na jednu z posledných predstavených techník

watermarking-u a navrhli sme novú techniku, ktorá je odolnejšia voči nedávno objaveným útokom.

Kapitola 1

Základné definície a pojmy

Watermarking je oblasť informatiky zaoberajúca sa vkladaním tajných správ do krycích správ. Tajnou správou, vodoznakom, býva spravidla informácia o autorských právach vzťahujúcich sa ku krycej správe. Cieľom vloženia vodoznaku je odradenie od kradnutia intelektuálneho vlastníctva a v prípade krádeže možnosť dokázať vlastníctvo alebo vystopovať osobu, ktorá začala neoprávnene šíriť kryciu správu – predmet ochrany.

Watermarking spustiteľného kódu sa zaoberá vkladaním vodoznakov do programov, a to tak, aby vodoznak mohol byť v programe spoľahlivo lokalizovaný a z neho extrahovaný (aj napriek útokom, ktoré sa tomu snažia zabrániť), aby bol v programe utajený a aby mal matematickú vlastnosť naznačujúcu, že jeho prítomnosť v programe je dôsledkom úmyselného činu.

Definícia 1.1. *Nech $\mathcal{P}$ je množina programov a $\mathcal{W}$ množina vodoznakov, $\mathcal{A}_\mathcal{V}$ množina pomocných vkladacích informácií a $\mathcal{A}_\mathcal{R}$ množina pomocných rozpoznávacích informácií.* ***Vkladajúca funkcia (vkladač)*** *je zobrazenie*

$$\mathcal{V} : (\mathcal{P} \times \mathcal{W} \times \mathcal{A}_\mathcal{V}) \to \mathcal{P}$$

a ***rozpoznávacia funkcia (rozpoznávač)*** *je zobrazenie*

$$\mathcal{R} : (\mathcal{P} \times \mathcal{A}_\mathcal{R}) \to \mathcal{W}$$

Pomocná vkladacia informácia $a_v \in \mathcal{A}_\mathcal{V}$ môže určovať časti programu, ktoré sa môžu, resp. nemôžu použiť pre vloženie vodoznaku. Taktiež može definovať hodnoty parametrov vkladania, napríklad počet častí, na ktoré je vodoznak rozdelený pred vkladaním.

Pomocná rozpoznávacia informácia $a_r \in \mathcal{A}_\mathcal{R}$ je nevyhnutná pre rozpoznanie vodoznaku v programe. Pomáha identifikovať štruktúry kódujúce vodoznak, napríklad ich adresy, veľkosti, topológiu, príp. iné vlastnosti.

Definícia 1.2. *Nech $\mathcal{V}$ je vkladač, $\mathcal{R}$ rozpoznávač. Usporiadanú dvojicu $(\mathcal{V}, \mathcal{R})$ nazývame* ***watermarking technika****.*

Za korektnú watermarking techniku $(\mathcal{V}, \mathcal{R})$ považujeme súbor postupov a algoritmov pre vloženie vodoznaku do programu a jeho rozpoznanie v tomto programe. Teda ak p je program, w vodoznak a a_v pomocná vkladacia informácia, tak v označenom programe p_w,

$$p_w = \mathcal{V}(p, w, a_v)$$

je vďaka pomocnej informácii a_r a rozpoznávaču $\mathcal{R}$ rozpoznateľný vložený vodoznak w,

$$w = \mathcal{R}(p_w, a_r)$$

Dvojicu (a_v, a_r) nazývame kľúč vodoznaku.

1.1 Kategórie watermarking techník

Techniky watermarking-u spustiteľného kódu delíme podľa

1. princípu uloženia a rozpoznávania vodoznaku
   a) statické – vodoznak je uložený v samotnom spustiteľnom súbore, rozpoznávanie analyzuje súbor bez jeho spustenia
   b) dynamické – vodoznak je uložený v dynamickom stave programu, jeho skonštruovanie a rozpoznanie vyžaduje spustenie programu
2. typu tajnej správy (informácie), ktorá je vodoznakom
   a) rýdzi vodoznak – najčastejšie názov držiteľa autorských práv a súvisiace informácie, výstupom rozpoznávania je vodoznak alebo pravdepodobnosť jeho výskytu v programe
   b) odtlačok – výsledkom procesu rozpoznavania je vodoznak, ktorý môže byť pre každú distribuovanú kópiu programu iný, t. j. identifikuje zákazníka
3. informácií potrebných pre rozpoznanie vodoznaku
   a) slepé – postačuje iba označený program alebo označený program a kľúč
   b) informované – vyžadujú aj neoznačený program a/alebo vodoznak samotný
4. etapy vkladania vodoznaku

   a) zdrojové – vkladanie vodoznaku do zdrojového kódu programu
   b) binárne – vkladanie vodoznaku do skompilovaného spustiteľného súboru

5. procesorovej architektúry

   a) natívny strojový kód – nerozlišuje sa medzi kódom a dátami, tieto sú netypové
   b) bajtkód virtuálneho stroja – striktne typový, obsahuje množstvo informácií (kvôli rôznym kontrolám), ktoré nie sú nevyhnutne potrebné pre beh programu

1.2 Požiadavky kladené na watermarking techniky

Aby mohla byť nejaká technika v praxi použitá, tak v závislosti od účelu, pre ktorý je nasadená, by mala spĺňať niektoré z nasledujúcich požiadaviek.

- **Zverejniteľnosť** – podobne ako pri šifrovaní, je samozrejmé požadovať zverejnenie použitých algoritmov, napr. aby nebolo pochýb o nezávislosti algoritmu a označeného programu pri rozpoznávaní pred súdom. Zverejnenie algoritmu taktiež umožňuje hľadanie a opravovanie jeho nedostatkov.
- **Dokázateľnosť** – pre vodoznak musí byť dokázateľné, že jeho prítomnosť nie je následkom náhody, ale úmyselného činu
- **Automatizácia** – proces vkladania a hlavne rozpoznávania vodoznaku musí byť automatizovaný, aby sa zabránilo podvádzaniu pri rozpoznávaní (zo strany dokazujúceho)

1.3 Kritéria kvality

Kvalitu každej z techník môžeme ohodnotiť podľa kapacity, odolnosti voči útokom a efektivity v zmysle nasledovných definícií.

Definícia 1.3. *Nech $\mathcal{P}$ je množina programov a $\mathcal{T}$ množina všetkých transformácií programov. Watermarking technika $(\mathcal{V}, \mathcal{R})$ je **odolná** voči transformácii $t \in \mathcal{T}$, ak pre všetky programy $p \in \mathcal{P}$ a pre všetky vodoznaky $w \in \mathcal{W}$ platí jedno z nasledujúcich:*

- $\mathcal{R}(t(p_w), a_r) = \mathcal{R}(p_w, a_r) = w$
- *efektivita a funkčnosť programu $t(p_w)$ je neakceptovateľne nízka*

- *vykonanie transformácie t na programe p_w je časovo neúnosné*

kde $p_w = \mathcal{V}(p, w, a_v)$ a (a_v, a_r) je kľúč vodoznaku.

Odolnosť techniky voči nejakej transformácii je teda úmerná schopnosti rozpoznávača rozpoznať vodoznak v transformovanom programe, časovej náročnosti vykonania transformácie a stratám na efektivite a funkčnosti transformovaného programu.

Definícia 1.4. ***Kapacita*** *techniky je priemerná hodnota podielu počtu bitov vodoznaku a počtu bajtov štruktúr kódujúcich vodoznak (kód a dáta) v programe.*

$$C = \frac{1}{|\mathcal{P}|.|\mathcal{W}|} \sum_{p \in \mathcal{P}, w \in \mathcal{W}} \frac{|w|}{|w_{code}| + |w_{data}|}$$

Veľkosť vložiteľného vodoznaku môže byť obmedzená veľkosťou programu, ak sa na zakódovanie vodoznaku používajú existujúce štruktúry programu.

Definícia 1.5. *Vodoznak w je* ***utajený*** *v programe p vzhľadom na štatistickú mieru programov M, ak je rozdiel $M(p) - M(p_w)$ zanedbateľný. Technika má* ***utajenosť*** *vzhľadom na štatistickú mieru M, ak sú vo všetkých programoch všetky vložiteľné vodoznaky utajené, t. j. ak pre techniku $(\mathcal{V}, \mathcal{R})$ platí*

$$\forall p \in \mathcal{P}: \ \forall w \in \mathcal{W}: \mathcal{V}(p, w, a_v) = p_w \implies M(p) - M(p_w) \quad \text{je zanedbateľné}$$

Utajenosť je teda miera nepriamo úmerná odlišnosti originálnych programov od označených programov.

Definícia 1.6. *Efektivita techniky je miera nepriamo úmerná*

1. *nárokom (časovým, pamäťovým, finančným) na vloženie a rozpoznanie vodoznaku, a to nárokom na:*
    - *vkladací a rozpoznávací softvér*
    - *užívateľov (watermarking techniky)*
2. *zvýšeným časovým a pamäťovým nárokom na vykonávanie označeného programu*

Podobne ako i v iných oblastiach, nie je možné maximalizovať všetky kritériá, lebo sa navzájom vylučujú. Napríklad programy označené odolnými technikami majú zväčša vyššie pamäťové a časové nároky, utajenosť môže byť v rozpore so snahou vložiť do programu čo najviac informácií (čo najväčší vodoznak) a pod. Preto je nutné určiť si pri výbere techniky priority a hľadať medzi jednotlivými kritériami vhodný kompromis.

1.4 Útoky voči watermarking technikám

Jedným z najdôležitejších kritérií kvality techník je odolnosť voči rôznym útokom. Útoky by mali samozrejme zachovávať sémantiku programu, až na straty, ktoré sme si „ochotní odpustiť“. Vo všeobecnosti predpokladáme, že každá technika je napadnuteľná manuálnym útokom pri investovaní veľkého množstva času na pochopenie fungovania programu. Preto nás zaujímajú hlavne automatizované útoky pomocou rôznych (hlavne softvérových) nástrojov. Z hľadiska operácií vykonaných na označenom programe delíme útoky do troch skupín[1]:

1.4.1 Odstránenie vodoznaku

Snahou útočníka je vodoznak odstrániť alebo dostatočne poškodiť, pri zachovaní postačujúcej funkcčnosti programu, tak, aby ho rozpoznávač nebol schopný rozpoznať, t. j. $\mathcal{R}(t(p_w), a_r) \neq w$. Vyžaduje to znalosť miest (statických alebo dynamických) v programe, ktoré kódujú vodoznak a schopnosť oklamať/odstrániť časti programu, ktoré majú zabrániť odstráneniu vodoznaku. Na odhalenie miest v programe, ktoré kódujú vodoznak, sa používajú napríklad tieto postupy:

1. optimalizácia – odstránenie mŕtveho kódu, ktorý by mohol obsahovať vodoznak
2. odhaľovanie štatistických anomálií (veľký rozdiel medzi $M(p)$ a $M(p_w)$ (def. 1.5))
   - staticky – distribúcia inštrukcií alebo výpočtov, napríklad použitie veľkých číselných konštánt
   - dynamicky – napríklad kód, ktorý sa nevykonáva kvôli nezrozumiteľnej podmienke[2], mnohokrát volané funkcie
3. porovnávanie kópií toho istého programu s rôznym vodoznakom (odtlačkom)[3]

1.4.2 Skreslenie programu

Aplikácia transformácií programu, ktoré zachovávajú užívateľom pozorovateľné správanie programu, s cieľom zabrániť rozpoznaniu vodoznaku, pri zachovaní postačujúcej kvality, t.

[1] angl. *subtractive*, *distortive* a *additive* útoky
[2] angl. opaque predicate
[3] angl. collusion attack

j. časových a pamäťových nárokov programu. Zmeny sa aplikujú rovnomerne na celý program alebo len na niektoré jeho časti, ak existujú indície hovoriace o umiestnení vodoznaku v programe. Skresľujúce útoky využívajú nasledovné techniky a ich kombinácie:

- dekompilácia a kompilácia
- optimalizácia – zarovnávanie kódu programu, použitie ekvivalentných inštrukcií a pod.
- obfuskácia – používa sa aj na ochranu pred útokmi (podrobnejšie v časti 1.5)
- iné – napríklad preusporiadanie častí programu, otáčanie podmienkových príkazov

Ak má útočník k dispozícií nástroj, príp. ďalšie potrebné informácie na rozpoznanie vodoznaku, môže sa snažiť malými zmenami programu dospieť do stavu, keď už rozpoznanie nebude úspešné. Hovoríme vtedy o útoku s *orákulom*[4] [20].

1.4.3 Pridanie ďalšieho vodoznaku

Útočník vloží do programu svoj vlastný vodoznak s cieľom zabrániť rozpoznaniu pôvodného vodoznaku alebo znemožniť rozhodnúť, ktorý vodoznak bol do programu vložený skôr, teda určenie, kto je skutočným tvorcom programu.

1.5 Ochrana proti útokom

Watermarking techniky využívajú nasledovné mechanizmy na zvýšenie odolnosti voči útokom. Používajú sa na ochranu voči jednotlivým útokom alebo celým triedam útokov alebo ich častí.

1. **Tamper-proofing** – ochrana proti zmene programu alebo jeho častí. Realizuje sa pridaním programového kódu, ktorý by mal odhaliť, či bol program zmenený, a po takomto zistení by mal spôsobiť čiastočnú alebo úplnú nefunkčnosť programu. Tamper-proofing sa používa na ochranu vložených vodoznakov, bezpečnostného kódu alebo na detekciu vírusovej infekcie. V princípe existujú tri spôsoby detekcie zmien programu

    - porovnanie programu s originálom pomocou kryptografickým hašovacích funkcií (MD5, SHA-1, SHA-256 atď.)
    - overenie správnosti medzivýsledkov programu

[4] angl. oracle attack

- generovanie programu počas jeho behu – aj malá zmena v generujúcom programe spôsobí pravdepodobne nefunkčnosť vygenerovaného programu [21]

2. **Obfuskácia (zahmlievanie)** – podľa [2] je to transformácia programu, ktorá

   - nezmení užívateľom pozorovateľné správanie
   - maximalizuje jeho nejasnosť, t. j. výrazne sťaží jeho pochopenie človekom, disassemblerom alebo dekompilátorom
   - maximalizuje jeho odolnosť voči deobfuskačným transformáciám – buď sa ich snaží znemožniť alebo spôsobiť časovo neefektívnymi
   - maximalizuje svoju utajenosť – zachováva štatistické vlastnosti programu
   - minimalizuje zvýšenie nákladov – zvýšené časové a pamäťové nároky transformovaného programu

   Primárnym cieľom obfuskácie je brániť útočníkovi v pochopení fungovania programu, resp. jeho algoritmov. Môže byť zacielená proti disassembleru, dekompilátoru alebo proti človeku. Jej význam stúpol po príchode softvéru distribuovaného vo forme architektonicky neutrálneho, ale ľahko dekompilovateľného a spätne-inžinierovateľného bytecodu (Java, .NET). Obfuskačné transformácie delíme podľa ovplyvňovaných subjektov:

   - Lexikálne charakteristiky – napríklad nahradenie zmysluplných mien premenných náhodnými
   - Riadenie toku – napríklad preusporiadavanie častí programu, nahradzovanie volaní funkcií ich telami a naopak, transformácie cyklov
   - Dáta – napríklad spájanie a rozdeľovanie premenných, preusporiadanie prvkov polí, generovanie statických dát procedúrami

   V oblasti watermarking-u spustiteľného kódu sa využíva na zahmlenie kódu alebo dát obsahujúcich vodoznak.

3. **Samoopravné kódy** – metóda pridania redundancie (nadbytočnej informácie) do vodoznaku alebo do štruktúr, ktoré ho reprezentujú. Umožňuje detekovať a opraviť isté množstvo chýb spôsobených napríklad skresľujúcim útokom. Ku príkladom patria *Hammingov kód*, *Reed-Solomon kód*, *Reed-Muller kód.*

Kapitola 2

Existujúce techniky

Táto kapitola je prehľadom najčastejšie spomínaných verejne známych a dostupných techník watermarking-u spustiteľného kódu.

2.1 Statické techniky

Sú to techniky, pri ktorých rozpoznanie vodoznaku prebieha statickou analýzou spustiteľného súboru, nevyžaduje sa jeho spustenie. Vodoznak môže byť uložený

- **v dátach** – statické inicializované dáta, v ktorých je vodoznak uložený nejakou technikou pre ukladanie vodoznakov v multimediálnych dátach (audio, video, fotografie, text)
- **v kóde** – princíp je analogický s vodoznakmi v oblasti multimédií, ktoré využívajú redundantnosť informácií vyplývajúcu z nedokonalosti ľudských zmyslov. Napríklad ľudské oko prakticky nevidí rozdiel vo farbe bodu, kódovaného trojicou farieb po 8 bitov, v ktorom sa najmenej významný bit každej z farieb líši od originálu. Každý takýto bod umožňuje zakódovanie troch bitov nejakej informácie [22]. V statickom kóde môžeme tento princíp pomenovať a definovať nasledovne:
  - usporiadanie – dva príkazy S_1 a S_2, medzi ktorými neexistujú závislosti, čo sa týka poradia vykonávania, usporiadame v lexikografickom poradí, ak je potrebné vložiť bit 1 a v opačnom poradí, ak je potrebné vložiť bit 0
  - ekvivalencia – definuje sa usporiadanie na množine ekvivalentných príkazov, jednotlivé príkazy vyberáme a používame v kóde podľa potrieb vodoznaku

  Rôzne techniky využívajú

- jednotlivé inštrukcie
- skupiny inštrukcií
- základné bloky
- moduly
- knižnice

Techniky založené na týchto princípoch sú zraniteľné pridaním ďalšieho vodoznaku – preusporiadaním príkazov alebo výmenou za ekvivalentné príkazy.

Vodoznak môže byť uložený aj v topológii grafu toku riadenia programu alebo vybraných inštrukcií.

Statické techniky sú vo všeobecnosti zraniteľné skresľujúcimi útokmi, najmä optimalizáciou a obfuskáciou. Podľa umiestnenia vodoznaku môže ísť o tieto transformácie:

- dátové – rozbíjanie dátových štruktúr na menšie časti a pridanie kódu na ich spojenie počas behu programu, nahradenie dát kódom, ktorý ich vytvorí po spustení programu
- kódové – vloženie nových vetiev do kódu alebo preusporiadanie existujúcich. Taktiež nie je problém vymeniť niektoré inštrukcie za ich ekvivalenty, zmeniť poradie inštrukcií (ak je to možné) atď.

2.1.1 Statické dáta

Popis. Najjednoduchšou technikou je vloženie nejakého textového reťazca do statickej dátovej sekcie spustiteľného súboru, napríklad reťazec `Jozko Mrkvicka (c) 2006`.
Zraniteľnosť. Takéto vodoznaky sa dajú obfuskáciou rozbiť na menšie kusy a tie rozdistribuovať po celom programe. Ďalším útokom je odstránenie všetkých dát statickej dátovej sekcie a pridanie kódu na ich vygenerovanie po spustení programu.

2.1.2 Digitálne média uložené v dátovej sekcií

Popis. Vodoznak je vložený v digitálnom médiu (obrázok, audio, video) nejakou technikou pre vkladanie vodoznakov do digitálnych médií. Takto označené médium sa nachádza v statickej dátovej sekcii spustiteľného súboru. Na techniku bol udelený patent v roku 1996 [10].

Zraniteľnosť. Proti už spomínaných útokom voči statickým dátam je možné použiť tamper-proofing – vložiť nepostrádateľnú časť programu medzi statické dáta a pri spustení ju extrahovať a spustiť. Avšak vytváranie a spúšťanie kódu počas behu programu je neštandardné a odhaliteľné správanie [1].

2.1.3 Usporiadanie základných blokov

Popis. Prvú skutočnú techniku watermarking-u spustiteľného kódu publikovali Davidson a Myhrvold [11], patentovanú ako US Patent 5 559 884 v septembri 1996. Vodoznak je uložený v usporiadaní základných blokov spustiteľného súboru.
Zraniteľnosť. Podobne, ako iné techniky založené na usporiadaní, aj táto je ľahko napadnuteľná náhodným preusporiadaním (základných blokov súboru).

2.1.4 Alokácia registrov

Popis. Na uloženie informácie do programu je možné využiť spôsob použitia všeobecných registrov procesora, ktoré sa líšia len menom. Využívame to, že daný algoritmus sa dá implementovať použitím rôznych registrov. Do toho, ktoré registre a v akom poradí ich používame, môžeme vložiť nejakú informáciu (vodoznak). Kapacita je závislá od procesorovej architektúry, v prípade RISC (veľa všeobecných registrov) je výrazne vyššia ako v prípade CISC (málo všeobecných registrov).
Zraniteľnosť. Podobne ako iné techniky založené na premenovávaní, aj táto je ľahko napadnuteľná. Napríklad použitím iných registrov, t. j. ďalším premenovaním. Použitie registrov ovplyvní aj optimalizácia a tiež dekompilácia a kompilácia programu.

2.1.5 Frekvencie skupín inštrukcií

Popis. Na začiatku vytvoríme vektor frekvencií výskytu vybraných skupín inštrukcií v programe. Vodoznak vložíme do vektora úpravou jeho koeficientov a následne upravujeme program tak, aby sme extrakciou vektora podľa prvej vety získali už upravený vektor. Úprava programu sa vykonáva nahradzovaním skupín inštrukcií ekvivalentnými skupinami. Výsledkom rozpoznávania je pravdepodobnosť, s ktorou je vodoznak v programe vložený a závisí od podobnosti (rôznosti) extrahovaného a očakávaného vektora.
Zraniteľnosť. Táto technika je napadnuteľná obfuskáciou a optimalizáciou kódu.

2.1.6 Nezrozumiteľné predikáty

Popis. V článku [7] opisuje Geneviève Arboit techniku pre watermarking Java programov, ktorá rozdelí vodoznak na k častí a každú z nich, spolu s indexom v rámci celého vodoznaku, zakóduje do nejakého nezrozumiteľného predikátu. Následne nájde v programe k podmienkových príkazov a podmienku každého z nich rozšíri o jeden z pripravených nezrozumiteľných predikátov. Rozpoznanie vodoznaku vyžaduje znalosť vkladaných nezrozumiteľných predikátov a výsledkom je pravdepodobnosť výskytu vodoznaku v programe.
Zraniteľnosť. Technika je napadnuteľná obfuskáciou a pridaním nezrozumiteľných predikátov [18].

2.1.7 Statická štruktúra behu programu

Popis. Program je abstrahovaný na graf, ktorého uzly tvoria základné bloky a hrany skoky a volania procedúr. Vodoznak je ďalší graf, ktorý spájame s grafom programu pridaním hrán medzi nimi. Techniku navrhol Venkatesan a kol. [19], analyzoval ju Collberg a kol. [6].
Zraniteľnosť. Technika je závislá na značení základných blokov programu, čo ju robí zraniteľnou voči skresľujúcim útokom.

2.2 Dynamické techniky

Vodoznak je uložený v stave (postupnosti stavov) bežiaceho programu, nie v samotnom kóde alebo statických dátach. Niektoré dynamické techniky sú preto ľahšie chrániteľné voči obfuskačným transformáciám pomocou tamper-proofing-u. Rozpoznanie vodoznaku vyžaduje spustenie programu, často so špeciálnym vstupom – kľúčom vodoznaku. Jednotlivé metódy sa líšia umiestnením vodoznaku v stave programu a v spôsobe jeho rozpoznania.

Vloženie vodoznaku prebieha na zdrojovom kóde alebo samotnom spustiteľnom súbore, teda vodoznak sa vkladá staticky do kódu alebo dát. Beh programu so špeciálnym vstupom interpretuje takto vložený vodoznak, ktorý je možné rozpoznať v stave (postupnosti stavov) programu – buď v dynamických dátach alebo vo vlastnostiach vykonaného kódu, a to bez ohľadu na spôsob a miesto statického uloženia.

2.2.1 Easter Egg

Popis. Easter Egg[1] je časť programu, ktorá sa vykoná len pri spustení programu s určitým neštandardným vstupom. Do tejto časti je nejakým spôsobom vložený vodoznak, napr. otvorí sa okno obsahujúce informáciu o autorských právach.
Zraniteľnosť. Takýto kód je ľahko lokalizovateľný (pomocou debuggera) a odstrániteľný.

2.2.2 Dynamické dáta

Popis. Vodoznak je skrytý v obsahu niekoľkých zvolených dátových štruktúr programu, tieto ho však obsahujú až po spustení programu s kľúčom vodoznaku. Rozpoznávanie prebieha pomocou na to určeného ku programu prilinkovaného kódu alebo pomocou debuggera. Výhodou tohto prístupu je, že ani po uhádnutí špeciálneho vstupu sa vykonanie programu navonok neodlišuje (program negeneruje žiadny výstup ako dôsledok jeho spustenia s kľúčom).
Zraniteľnosť. Efektívnym útokom je obfuskácia, napr. spájanie a rozdeľovanie premenných znemožní rozpoznávaču určiť hodnoty potrebných premenných.

2.2.3 Dynamic execution trace

Popis. Táto technika ukladá vodoznak do postupnosti vykonaných inštrukcií alebo „navštívených“ adries počas behu programu na špeciálnom vstupe. Vodoznak je rozpoznávaný sledovaním nejakej (potenciálne štatistickej) vlastnosti postupnosti adries alebo postupnosti vykonaných inštrukcií (alebo operácií).
Zraniteľnosť. Technika je napadnuteľná obfuskáciou a optimalizáciou, ktoré ovplyvňujú navštívené adresy a použité inštrukcie v programe. Podobne ako pri statických vodoznakoch uložených v kóde je možné nahradiť inštrukcie/základné bloky ich ekvivalentami.

2.2.4 Topológia dynamicky vybudovaného grafu

Popis. Collberg a Thomborson [1] navrhli známu techniku, ktorá ukladá vodoznak do grafu vytvoreného počas behu programu. Správne vytvorenie grafu a následné rozpoznanie vodoznaku vyžaduje spustenie programu so špeciálnym vstupom (kľúčom vodoznaku).
Zraniteľnosť. Odolnosť techniky spočíva v náročnosti analýzy kódu, ktorý spravuje graf, teda v náročnosti vytvoriť sémantiku zachovávajúce transformácie programu, ktoré pod-

[1] Informácie o známych Easter Egg vodoznakoch z oblastí multimédií a softvéru sú verejne prístupné na internetovej stránke http://www.eeggs.com/.

```
1: if G_P then
2:     ...
3:     x ← vyraz
4:     ...
5: end if
```

Obr. 2.1: Priradenie do premennej chránené podmienkou G_P (Guard Predicate)

statne zmenia topológiu grafu. Zároveň je ľahšie chrániť takýto vodoznak pomocou tamper-proofing ako napríklad samotný kód. Pre veľké množstvo programov je však táto technika nevhodná, lebo kód manipulujúci s grafom sa nehodí medzi ostatné časti programu (def. 1.5) a teda je ľahké ho objaviť.

2.2.5 Watermarking použitím výrezov programov (SWuS)

Popis: Výrez programu je program skladajúci sa z tých príkazov programu, ktoré priamo alebo nepriamo ovplyvňujú hodnoty zvolenej podmnožiny premenných v nejakom bode programu.

Technika SWuS [5] skrýva vodoznak do stavu výrezu, ktorý je nedosiahnuteľný počas normálneho behu programu, a je odhalený počas rozpoznávania vodoznaku.

Technika má tri fázy:

1. vloženie vodoznaku do výrezu S premennej x a bodu K
2. rozpoznanie premennej x, bodu K a výrezu S, jeho úprava dopredu pripravenou množinou zmien
3. konštrukcia vodoznaku vykonaním upraveného programu

Prvá fáza. Zvolíme premennú x a výrez S, v rámci ktorého sa nachádza priradenie do premennej x (bod K), ktoré sa vykoná len po splnení nejakej podmienky (obr. 2.1).

Následne zvolíme nezrozumiteľnú podmienku O_P a pridáme ju k podmienke G_P tak, aby neovplyvnila význam (obr. 2.2).

Teraz nahradíme priradenie do premennej x dvoma priradeniami tak, aby konečná hodnota x bola totožná s prípadom jednoduchého priradenia (obr. 2.3).

Funkcie F_1 a F_2 (obr. 2.4) zostrojíme nasledovne:

- P_i platí práve vtedy, keď platí Q_i (pre $i \in \{1, \ldots, n-1\}$)
- ak neplatí G_P, tak neplatí žiadna z podmienok P_i, resp. Q_i

1: **if** G_P **and** O_P **then**
2: ...
3: $x \leftarrow vyraz$
4: ...
5: **end if**

Obr. 2.2: Rozšírenie o podmienku O_P s pravdivostnou hodnotou **true**

1: **if** G_P **and** O_P **then**
2: ...
3: $x \leftarrow F_1$
4: $x \leftarrow x + F_2$
5: ...
6: **end if**

Obr. 2.3: Nahradenie priradenia dvojicou priradení

- ak platí G_P (riadenie programu sa dostane aj do funkcií F_1 a F_2), tak platí aspoň jedna z podmienok P_i funkcie F_1, resp Q_i funkcie F_2. Znamená to tiež, že riadenie sa nikdy nedostane k poslednej **else** vetve vo funkcii F_1, a že sa vykoná práve jeden príkaz A_i a práve jeden príkaz B_i.

Prislúchajúce dvojice príkazov A_i a B_i musia zabezpečiť, aby výsledná hodnota premennej x v hlavnom programe bola vždy rovnaká.

Funkcie sú podobné, avšak líšia sa v poslednej vetve – vykonanie A_n v F_1 je podmienené platnosťou P_n, vo funkcii F_2 vykonanie B_n nie je podmienené. Práve tu je miesto pre vodoznak – v návratovej hodnote B_n. Pri normálnom vykonaní programu, ak sa vykoná B_n, tak sa predtým muselo vykonať A_n, čo efekt B_n neutralizovalo do podoby pôvodného priradenia.

Druhá fáza. Počas rozpoznávania, po identifikácii premennej x, podmienkového príkazu

if QP **and** OP

a priradení, upravíme program tak, aby výraz podmienkového príkazu bol pravdivý aj vtedy, ak žiadna z podmienok P_i vo funkcii F_1 neplatí, napríklad pridaním **or True** (obr. 2.5).

Tretia fáza. Po tejto úprave bude existovať vstup programu, po ktorom by v pôvodnom programe výraz podmienkového príkazu neplatil, ale teraz platiť bude. Zároveň vo funkcii

```
FUNCTION F_1
 1: if P_1 then
 2:     A_1
 3: else if P_2 then
 4:     A_2
 5:     ...
 6: else if P_n then
 7:     A_n
 8: else
 9:     return 0
10: end if
FUNCTION F_2
 1: if Q_1 then
 2:     B_1
 3: else if Q_2 then
 4:     B_2
 5:     ...
 6: else
 7:     B_n
 8: end if
```

Obr. 2.4: Pomocné funkcie F_1 a F_2

```
1: if (G_P and O_P) and TRUE then
2:     ...
3:     x ← F_1
4:     x ← x + F_2
5:     ...
6: end if
```

Obr. 2.5: Úprava podmienkového príkazu

F_1 neplatí ani jedna z podmienok P_i a teda funkcia vráti 0. Avšak vo funkcii F_2 sa vykonajú príkazy bloku B_n, teda nakoniec bude v premennej x návratová hodnota z B_n, čo je hľadaný vodoznak.

Kľúčom vodoznaku sú informácie potrebné pre lokalizáciu príslušných premenných, podmienkových príkazov a príkazov priradenia, ktoré sú výstupom vkladania a vstupom rozpoznávania vodoznaku.

Zraniteľnosť. Takto predstavená technika je zraniteľná niektorými skresľujúcimi útokmi, avšak jej autori poskytujú riešenia na konkrétne problémy. Taktiež používajú samoopravný Reed-Solomon kód na všeobecné zvýšenie odolnosti. SWuS technika je však zraniteľná pridaním ďalšieho vodoznaku, po ktorom sa nedá určiť, ktorý vodoznak originálny.

2.2.6 Dynamic Path-Based Watermarking (DPB)

Úvod

Základnou myšlienkou tohto prístupu je vloženie vodoznaku do štruktúry vetvenia programu. Má to niekoľko výhod

- vetvy programu sú to, čo robí program jedinečným, nedajú sa teda len tak odstrániť
- nachádzajú sa v celom a v každom zmysluplnom programe, takže označený program by nemal byť štatistickou anomáliou
- vetva programu sa vykoná alebo nevykoná – je teda binárna a môže dobre slúžiť pre uloženie jedného bitu informácie

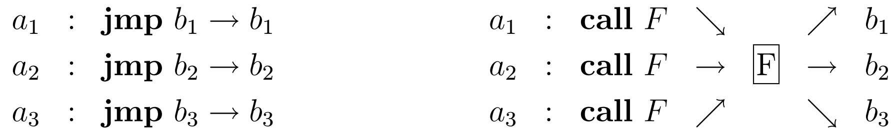
Na uvedenom princípe je možné vybudovať viacero techník, radia sa medzi dynamické, slepé a odtlačkové. Jedna z nich bola prezentovaná [4] vo verziách pre architektúry Java a IA32 . V tejto práci sa venujeme verzii pre architektúru IA32.

Popis

Technika je postavená na vetviacich funkciách v zmysle nasledujúcej definície.

Definícia 2.1. *Vetviaca funkcia je funkcia, ktorá po skončení vykonávania svojho tela presmeruje beh programu na iné miesto ako to, z ktorého bola zavolaná.*

Pri každom volaní si vetviaca funkcia vypočíta novú návratovú adresu. Vstupom tohto výpočtu je okrem iného aj pôvodná návratovej adresa.



Obr. 2.6: Použitie vetviacej funkcie

Vetviaca funkcia môže byť použitá na realizáciu skupiny nepodmienených skokov (obr. 2.6).

Výpočet novej návratovej adresy realizujeme nasledovne. Pripravíme tabuľku, ktorej obsahom budú nové návratové adresy. Index správnej adresy získame z pôvodnej návratovej adresy pomocou perfektného hašovania.

$$\text{ret-address}_{temp} \leftarrow T(h(\text{ret-address}_{original}))$$

Nové návratové adresy však budú kvôli utajeniu *xorované* pôvodnými adresami, takže na adresu z tabuľky aplikujeme

$$\text{ret-address}_{new} \leftarrow \text{ret-address}_{temp} \ \mathbf{XOR} \ \text{ret-address}_{original}$$

V rámci inštrukcie skoku môžme zakódovať jeden bit informácie napr. ako výsledok porovnania parity zdrojovej a cieľovej adresy alebo porovnania veľkosti zdrojovej a cieľovej adresy. Vetviaca funkcia reprezentujúca niekoľko inštrukcií skoku môže týmto spôsobom zakódovať celý reťazec bitov – vodoznak.

Nasledujúce časti opisujú variant založený na usporiadaní zdrojovej a cieľovej adresy.

Vloženie vodoznaku

V programe zvolíme inštrukciu nepodmieneného skoku. Označme jej adresu l_{begin} a adresu jej cieľa l_{end}. Túto inštrukciu nahradíme volaním vetviacej funkcie, ktorá potom presunie riadenie programu na adresu a_1, ktorá bude menšia ako l_{begin}, ak potrebujeme vložiť bit 1, alebo väčšia, ak potrebujeme vložiť bit 0. Potom na a_1 vložíme volanie vetviacej funkcie, ktorá presunie riadenie na a_2, pričom a_2 bude menšie ako a_1, ak ... atď. Adresy a_i samozrejme nemôžeme vyberať ľubovoľne, t. j. volania vetviacej funkcie nemôžeme vkladať na ľubovoľné miesta v programe. Nutnou podmienkou je, aby sa k nim nedostalo riadenie pôvodného programu, t. j. pred inštrukciou volania vetviacej funkcie na adrese a_i sa musí nachádzať nepodmienený skok (obr. 2.7).

Nová návratová adresa vetviacej funkcie, po jej zavolaní z adresy a_n, je cieľová adresa l_{end}. Kľúčom vodoznaku je dvojica adries l_{begin} a l_{end}.

$$a_i - c \;:\; \textbf{jmp } \textit{niekam} \qquad a_i \;:\; \textbf{call } f$$

c je počet bajtov inštrukcie: **jmp** *niekam*

Obr. 2.7: Volanie vetviacej funkcie

Rozpoznanie vodoznaku

Spustíme program a krokujeme ho medzi adresami l_{begin} a l_{end}. V tejto stope rozpoznáme vetviacu funkciu (po jej skončení sa vykonávanie programu nevráti na inštrukciu nasledujúcu volanie vetviacej funkcie) a analyzujeme adresy, na ktoré vracia riadenie programu. Vzájomným porovnávaním adries získame jednotlivé bity vodoznaku.

Používanie absolútnych adries vetviacou funkciou pri mapovaní a_i na a_{i+1} zabezpečuje ochranu proti pridaniu vodoznaku a skresľujúcim útokom. Na ochranu proti odstráneniu sa používa tamper-proofing – súčasťou vetviacej funkciu je kód, ktorého odstránenie (v rámci odstránenia celej vetviacej funkcie) spôsobí nefunkčnosť samotného programu, napr. úprava cieľa nejakého skoku.
Zraniteľnosť. Matias Madou a kol. [9] opisuje úspešný útok založený na

- vybudovaní grafu toku riadenia pomocou analýzy behu programu a statického disassemblovania
- odhalení návratových adries a dát zapisovaných tamper-proofing kódom vetviacej funkcie
- odstránení vetviacej funkcie a jej volaní, nahradenie kódom simulujúcim tamper-proofing kód

Ich analýza tiež odhalila vysoký počet volaní vetviacej funkcie, ktorý je podozrivý a má za následok jednoduchšie odhalenie vetviacej funkcie.

2.3 Hybridné techniky

2.3.1 Abstraktný watermarking

Popis: Technika [3] patrí čiastočne do oboch kategórií techník – statických aj dynamických. Vkladanie vodoznaku sa skladá z dvoch častí

1. vytvorenie programu p', v ktorého sémantike je ukrytý vodoznak w

2. vloženie programu p' do hlavného programu p – vytvorenie p_w

Označený program p_w nie je potrebné kvôli rozpoznaniu vodoznaku spúšťať, preto by sme mohli túto techniku zaradiť medzi statické. Zároveň však je vodoznak ukrytý v konkrétnej sémantike programu a rozpoznávame ho abstraktnou interpretáciou tejto sémantiky. Môžeme ju teda zaradiť aj medzi dynamické techniky.

Zraniteľnosť: Vodoznak je ukrytý v hodnotách, ktoré sú priradzované premenným. Techniku je možné napadnúť obfuskáciou:

- rozdelenie premenných – rozpoznávač nebude schopný lokalizovať potrebné premenné
- obfuskáciou výpočtov – rozpoznávač nebude schopný určiť hodnoty, ktoré sú do týchto premenných priradené
- pridanie mŕtvych vetiev s priradením do premennej – pribudnú priradenia, ktoré nesúvisia s vodoznakom

2.4 Zhrnutie

V tejto kapitole sme predstavili väčšinu existujúcich techník. Hoci sú všetky z nich zraniteľné voči nejakému útoku, líšia sa v náročnosti jeho vykonania, čo umožňuje ich porovnávanie z hľadiska odolnosti (1.3).

Zraniteľnosť techník závisí od množstva informácií, ktoré má útočník k dispozícii. Neznalosť použitej techniky teoreticky nebráni jej napadnutiu, ak je zraniteľná skresľujúcim útokom, lebo útočník môže aplikovať všetky známe relevantné skresľujúce útoky. V praxi je však takýto útok nerealizovateľný kvôli jeho nízkej efektivite, najmä časovej náročnosti a zníženiu výkonu napadnutého programu.

Ak útočník pozná použitú statickú techniku, tak je v jednoduchej situácii, lebo tieto techniky sú zväčša napadnuteľné jednoduchými automatizovanými útokmi. Preto sa im v dnešnej dobe už nevenuje veľa pozornosti. Spomenuté dynamické techniky sú tiež napadnuteľné, ale už vyžadujú istú mieru heuristiky. Nespĺňajú teda podmienku zverejniteľnosti. Priznáva sa im však väčší potenciál a sú predmetom výskumu.

Znalosť použitej techniky a kľúča vodoznaku činí útok ešte jednoduchším:

- **SWuS** (2.2.5) – kľúčom je informácia postačujúca pre nájdenie premennej, podmienkového príkazu a priradenia do premennej, čo umožňuje nájsť príslušnú nedosiahnuteľnú časť kódu a odstrániť ju

- **DPB** (2.2.6) – znalosť kľúča odbremení útočníka od hľadania vetviacej funkcie; následne stačí použiť už spomenutý útok
- **Abstraktný watermarking** (2.3.1) – znalosť premenných, ktoré sú použité pre vloženie vodoznaku, umožňuje presnejšie zacieliť obfuskáciu, ktorá zabráni rozpoznaniu vodoznaku

Ak je k dispozícii kľúč vodoznaku a rozpoznávač, avšak nie je známa použitá technika, jedná sa o útok s orákulom. Útočník postupne aplikuje známe útoky na všetky známe techniky. Keďže je každá transformácia vykonaná na pôvodnom súbore, výkon programu nebude tak ovplyvnený, ako v prípade, keď nemáme žiadne informácie a musíme aplikovať všetky transformácie súčasne.

Kapitola 3

Návrh watermarking techniky

Prirodzenou cestou pri návrhu novej techniky bolo vychádzať z nejakej existujúcej. Za základ sme vybrali techniku DPB (2.2.6). Viedli nás k tomu nasledovné dôvody:

1. DPB patrí medzi dynamické techniky, ktoré sú vo všeobecnosti považované za sľubnejšie, čo sa týka odolnosti. Statické sú považované „slabé“ a už nie sú prakticky predmetom skúmania.

2. DPB je relatívne nová technika, podobné techniky sú málo prebádané a je preto vhodným odrazovým mostíkom pre ďalšie skúmanie v tejto oblasti

3. Princíp DPB, ktorým je uloženie vodoznaku do grafu toku riadenia, je tiež pomerne nový a získal si naše osobné preferencie

4. Úspešné prelomenie DPB – v novembri roku 2005 vydali Matias Madou a kol. článok [9] opisujúci úspešné prelomenie tejto techniky. Existuje preto prirodzená snaha o vývoj odolnejšej techniky využívajúcej dobré myšlienky DPB.

Nasledujúca časť opisuje a analyzuje úspešný útok, ktorý už bol v spomenutý v časti 2.2.6.

3.1 Hybridný staticko-dynamický útok

Základom útoku [9] je konštrukcia grafu toku riadenia programu, ktorého vrcholmi sú inštrukcie programu a hranami prechody riadenia medzi inštrukciami[1].

[1] v celej práci používame tu uvedenú definíciu, v klasickej definícii grafu toku riadenia sú vrcholmi základné bloky programu

3.1.1 Konštrukcia grafu toku riadenia

Konštrukcia grafu toku riadenia vyžaduje korektnú identifikáciu inštrukcií a prechodov riadenia v programe.

Základným prostriedkom pre získanie inštrukcií je disassemblovanie programu. Úplné statické zdisassemblovanie programov pre počítače Von Neumannovskej architektúry však nie je vo všeobecnosti dosiahnuteľné, lebo rozlíšenie dát a inštrukcií je nerozhodnuteľným problémom. Pre kód vygenerovaný bežnými kompilátormi je tento problém relatívne úspešne riešiteľný. Jedným z prostriedkov je použitie dynamickej inštrumentácie[2] na zaznamenanie všetkých vykonaných inštrukcií (spolu s adresami) programu počas jeho behu, napr. pomocou nástroja DIOTA [12]. Základným princípom je nechať bežiaci program nezmenený v pamäti a priebežne generovať inštrumentovaný kód. Vygenerovaný kód bude skonštruovaný tak, aby použil originálny program pre prístupy dátam a vygenerovaný program pre prístupy do kódu, t. j. počas behu sa používa kód z vygenerovaného klonu a dáta z originálneho programu. Týmto spôsobom je možné dosiahnuť asi 80 percentné pokrytie [14]. Väčšina zvyšných inštrukcií sa dá identifikovať statickým rekurzívnym disassemblovaním.

Z identifikovaných inštrukcií sa zostrojí graf toku riadenia, ktorý je nadgrafom skutočného grafu toku riadenia, lebo obsahuje zbytočné hrany. Je to dôsledkom nedostatku informácií o niektorých vrcholoch – inštrukciách. Napríklad konzervatívny prístup k vyhodnoteniu nepriamych skokov spôsobí, že takýto skok môže mať cieľ na ľubovoľnom mieste programu, pričom v skutočnosti je cieľov iba niekoľko. Podobne predpoklad, že podmienka podmieneného skoku môže byť pravdivá aj nepravdivá, je nesprávny v prípade použitia nezrozumiteľnej podmienky, ktorá má konštantnú pravdivostnú hodnotu. Dynamická inštrumentácia však zaznamená nielen inštrukcie, ale aj skutočne použité hrany grafu toku riadenia.

Výsledkom je graf, ktorý oproti skutočnému grafu toku riadenia obsahuje podmnožinu hrán medzi vrcholmi v časti, ktorá je získana vďaka dynamickej inštrumentácii (pretože nie všetky skutočné hrany sa podarí zaznamenať), a nadmnožinu hrán medzi vrcholmi pokrytými statickým disassemblovaním (pretože tu sme v rovnakej situácii ako vo vyššie spomínanom nadgrafe). Tento graf je zväčša podobnejší skutočnému, než predtým získaný nadgraf.

[2]techniky používané na zmenu existujúcich programov za účelom zberu dát počas ich vykonávania

3.1.2 Možnosti útoku

Znalosť grafu toku riadenia[3] programu umožňuje

- odhaliť odchýlky v štatistických vlastnostiach programu (počet miest volania funkcie, hĺbka volaní funkcií atď.)
- odhaliť kód, ktorý nie je vygenerovaný kompilátorom (inštrukcie, skupiny inštrukcií atď., ktoré kompilátory nezvyknú generovať)
- zmeniť časti programu (grafu toku riadenia)[4]
  - vykonať analýzu toku dát a následnú optimalizáciu
  - inak manuálne alebo poloautomaticky zmeniť program pri zachovaní vstupno-výstupnej sémantiky

Okrem konštrukcie grafu toku riadenia programu, umožňuje inštrumentácia aj detekovať a následne analyzovať skutočne vykonaný kód – sled v grafe toku riadenia. Napríklad vieme zistiť, koľkokrát bola nejaká funkcia vykonaná. Navyše poznáme aj skutočné hodnoty premenných a priradenia do nich.

3.1.3 Aplikácia útoku na DPB

V spomínanom článku [9] autori uvádzajú, že aj bez znalosti použitej techniky je možné, vďaka konštrukcii grafu toku riadenia, odhaliť nasledovné:

1. nezvyčajne veľký počet miest volania jednej funkcie
2. funkciu meniacu svoju návratovú adresu

Následná analýza podozrivej funkcie odhalila, že po jej zavolaní sa zmenia tieto časti stavu programu:

- návratová adresa funkcie
- dve pamäťové miesta v dátovej sekcii programu (tamper-proofing ochrana)

[3]tu, ako aj v ďalšom texte, máme na mysli graf, ktorý je možné zostrojiť postupom z časti 3.1.1, nie skutočný graf toku riadenia

[4]závislé od kvality zostrojeného grafu

S použitím debuggera odhalili vstupno-výstupný vzťah funkcie (návratová adresa ver. nová návratová adresa a dve zmenené pamäťové miesta). Následne odstránili vetviacu funkciu a nasimulovali jej správanie (tamper-proofing ochranu) inou funkciou.

Z uvedeného vyplývajú hlavné nedostatky DPB techniky:

- prítomnosť vodoznaku a umiestnenie jeho štruktúr sú odhaliteľné štatistickou analýzou programu
- vodoznak je implementovaný jedinou funkciou, ktorej poškodenie znamená úspešné napadnutie (bod zlyhania[5])
- technika je v podstate jednoduchá a zraniteľná poloautomatizovaným útokom

3.1.4 Návrh novej techniky

Pri návrhu novej techniky sa musíme vyrovnať s opísanými možnosťami hybridného staticko-dynamického útoku. Pri všeobecnom pohľade sa naskytajú takéto možnosti:

- Zabrániť konštrukcii grafu toku riadenia
  - zmenšiť podiel kódu a ciest v kóde, ktoré je realistické spracovať pomocou dynamickej inštrumentácie
  - obfuskácia zacielená proti statickým disassemblerom
- Zabrániť detekcii kľúčových štruktúr vodoznaku
  - vloženie vodoznaku nesmie výrazne zmeniť štatistické vlastnosti programu (počet funkcií, počet volaní funkcie atď.)
  - kód by mal byť vygenerovateľný nejakým kompilátorom – neobsahuje inštrukcie, skupiny inštrukcií a pod., ktoré kompilátory nepoužívajú, napr. zmena návratovej adresy funkcie
- Zabrániť odstráneniu kľúčových štruktúr vodoznaku
  - obfuskácia
  - odstrániť bod zlyhania
  - rôznorodosť – tým istým spôsobom sa nedajú odstrániť všetky (ani len dve) časti vodoznaku

[5] angl. Single Point of Failure – prvok systému, ktorý pri výpadku spôsobí jeho nefunkčnosť

3.2 Základný princíp novej techniky

S predošlou časťou na zreteli, sme navrhli novú techniku watermarking-u spustiteľného kódu, vychádzajúcu z DPB techniky. Základným princípom je uloženie vodoznaku do grafu volania funkcií alebo do sledov nad týmto grafom. Vrcholmi grafu sú funkcie, hrany sú volania medzi funkciami. Je to teda podgraf grafu toku riadenia programu.

Funkcie kódujúce vodoznak (ďalej len WM funkcie) sú nové funkcie pridané do programu. Každá počíta nejaký pre program užitočný výraz. Pri vkladaní je kód takéhoto výrazu nahradený volaním WM funkcie. Vloženie vodoznaku do programu prebieha v nasledovných krokoch:

1. Rozdelenie vodoznaku na úseky a ich zakódovanie do grafov (sledov)
2. Výber vhodných výrazov pre WM funkcie
3. Vytvorenie a vloženie WM funkcií do programu

   (kompilácia programu)

4. Doplnenie potrebných dát do skompilovaného programu

Kľúčom vodoznaku sú informácie potrebné pre identifikáciu WM funkcií. Rozpoznávanie vodoznaku prebieha nasledovne:

1. Konštrukcia grafu volania funkcií (a sledov nad ním)
2. Rozpoznanie WM funkcií vďaka pomocnej informácii
3. Konštrukcia úsekov vodoznaku z grafov volania WM funkcií
4. Spojenie úsekov do hľadaného vodoznaku

Jednotlivé kroky sa dajú implementovať viacerými spôsobmi. Naša technika nepredpisuje, ktorý spôsob použiť. Definuje základný rámec, opisuje možnosti a ich vlastnosti a necháva užívateľovi slobodu pre výber tých, ktoré sú pre jeho potreby najvhodnejšie.

3.3 Vloženie vodoznaku do programu

3.3.1 Rozdelenie vodoznaku

Jedným zo spôsobov zvýšenia odolnosti techniky je decentralizácia uloženia vodoznaku v programe, s cieľom odstránenia tzv. bodu zlyhania – možnosti zničiť celý vodoznak

zničením jednej jeho časti. Základným nástrojom pre dosiahnutie tohto cieľa je rozdeliť vodoznak (t. j. reťazec bitov) na viac častí (úsekov) a každú z nich zakódovať do programu samostatne tak, aby napadnutie jednej časti programu spôsobilo neschopnosť rozpoznať len malý počet úsekov vodoznaku, pričom ostatné úseky ostanú neporušené.

Implementácia vyžaduje rozhodnúť o hodnotách nasledovných parametrov:

- dĺžka úsekov – konštantná alebo variabilná
- počet úsekov – veľký počet zvyšuje množstvo informácie potrebnej na usporiadanie úsekov, malý počet znamená veľké úseky, a teda väčšie straty pri zničení jedného úseku
- prekrývanie úsekov – prekrývanie zvýši odolnosť pri čiastočnom poškodení, zníži však kapacitu
- spôsob očíslovania úsekov – ak sú úseky kódované úplne nezávisle (pozri 3.4.2), je nutné ich nejakým spôsobom očíslovať, aby ich bolo možné pri rozpoznaní usporiadať
  - úsek obsahuje svoje ID; v prípade neprekrývajúcich sa úsekov potrebujeme zakódovať `počet úsekov` úsekov dĺžky
    $$\frac{\texttt{dĺžka vodoznaku}}{\texttt{počet úsekov}} + \log_2 \texttt{počet úsekov}$$
    teda spolu
    $$\texttt{dĺžka vodoznaku} + \texttt{počet úsekov} * \log_2 \texttt{počet úsekov}$$
    bitov
  - ID úseku je ID WM funkcie, v ktorej úsek začína; úsekov môže byť menej ako WM funkcií
  - úsek obsahuje ID bitu, od ktorého kóduje vodoznak; v prípade neprekrývajúcich sa úsekov potrebujeme zakódovať `počet úsekov` úsekov dĺžky
    $$\frac{\texttt{dĺžka vodoznaku}}{\texttt{počet úsekov}} + \log_2 \texttt{dĺžka vodoznaku}$$
    teda spolu
    $$\texttt{dĺžka vodoznaku} + \texttt{počet úsekov} * \log_2 \texttt{dĺžka vodoznaku}$$
    bitov
  - ID WM funkcie, v ktorej úsek začína, je ID bitu, od ktorého úsek kóduje vodoznak, t. j. počet funkcií je zhodný s počtom bitov vodoznaku

3.3.2 Kódovanie (úsekov) vodoznaku

Základným princípom je zakódovanie reťazca bitov do topológie grafu nad funkciami a ich vzájomnými volaniami. Uvažujeme len zakorenené orientované grafy, v ktorých existuje aspoň jedna cesta z koreňa do každého vrchola. Vhodnou podmienkou je acyklickosť, lebo cykly s dĺžkou viac ako jeden (rekurzívne funkcie) sa v programoch prakticky nevyskytujú, a preto by boli podozrivé. Orientáciu hrán môžeme, ale nemusíme pri kódovaní informácie využiť.

Vloženie informácie vyžaduje definovanie usporiadania na grafoch, aby sme mohli pre každý graf určiť, čo kóduje, a naopak, aby sme pre kódované číslo vedeli zostrojiť príslušný graf. Na kódovanie informácií môžeme použiť len vhodne zvolenú podmnožinu všetkých potenciálnych grafov, lebo určenie čísla na základe grafu je vo všeobecnosti ťažký problém (problém izomorfizmu grafov [13]).

Neoznačené grafy. Nevyžadujú usporiadanie vrcholov, a preto majú nízku kapacitu. Príkladom sú všeobecné stromy, kapacita v príklade z [1] je $n \log_2 2.956$ bitov pre n vrcholov.

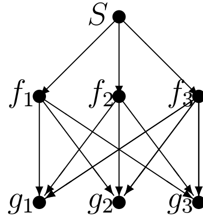
Označené grafy. Vyžadujú definovanie usporiadania (t. j. pomenovania, resp. rozlíšenia) vrcholov – WM funkcií. Rozlišujeme

- implicitné usporiadanie, napr. na základe
  - adries funkcií v programe
  - veľkosti kódu funkcií
  - poradia pri vykonávaní programu
- explicitné usporiadanie, napr.
  - kód WM funkcie obsahuje jej ID

Označené grafy majú vyššiu kapacitu než neoznačené, vo všeobecnosti sú však zraniteľné voči útokom, ktoré menia usporiadanie vrcholov.

Na rozdiel od matematických grafov, ktoré sú statické, sú grafy nad WM funkciami dynamicky skonštruované ich prechodom, ktorý v podstate definuje usporiadanie na vrcholoch. Je to vlastne sled obsahujúci všetky vrcholy a hrany. Preto sú grafy nad WM funkciami označené (majú vysokú kapacitu) a zároveň sú odolné voči zmene usporiadania, lebo zmena poradia volania WM funkcií mení sémantiku programu (za predpokladu vhodnej konštrukcie WM funkcií).

Jeden z možných spôsobov kódovania vodoznaku využíva princíp zobrazený na obrázku 3.1. Ak máme k dispozícii m WM funkcií, tak tri z nich použijeme ako pomocné (S, A, B)



Obr. 3.1: Usporiadanie vrcholov prechodom grafu

a zvyšné funkcie rozdelíme do r riadkov. Každý bit je kódovaný podľa toho, či príslušná cesta z S končí v A alebo v B. Cesty sú lexikograficky usporiadané. Cesta

$$S, f_{r0_{a_0}}, f_{r1_{a_1}}, \dots, f_{r_{n_{a_n}}}, A$$

znamená, že bit

$$a_0.|r_1|.|r_2|.\dots.|r_n| + a_1.|r_2|.|r_3|.\dots.|r_n| + \dots + a_n$$

je 0, podobne cesta

$$S, f_{r0_{a_0}}, f_{r1_{a_1}}, \dots, f_{r_{n_{a_n}}}, B$$

znamená, že bit

$$a_0.|r_1|.|r_2|.\dots.|r_n| + a_1.|r_2|.|r_3|.\dots.|r_n| + \dots + a_n$$

je 1 (r_i je i-ty riadok, $|r_i|$ je počet funkcií v i-tom riadku, a_i je index funkcie v riadku, n je počet riadkov).

Pre maximalizáciu kapacity potrebujeme maximalizovať počet ciest z S do vrcholov A alebo B. Pre m funkcií to znamená určiť hodnoty a počet čísel, ktorých súčet je m tak, aby ich súčin bol maximálny. Z AG nerovnosti vyplýva, že na dosiahnutie maximálneho súčinu musia byť hľadané čísla rovnaké (resp. líšiace sa maximálne o jedna v prípade prirodzených čísel). Ak x je hľadaný počet čísel, tak ich súčin je $\left(\frac{m}{x}\right)^x$. Nájdime teda maximum funkcie $\left(\frac{m}{x}\right)^x$:

$$\left(\left(\frac{m}{x}\right)^x\right)' = \left(e^{x.\ln\frac{m}{x}}\right)' = e^{x.\ln\frac{m}{x}}.\left(\ln\frac{m}{x} + x.\frac{1}{\frac{m}{x}}.m.\left(-\frac{1}{x^2}\right)\right) = \left(\frac{m}{x}\right)^x\left(\ln\frac{m}{x} - 1\right)$$

$$\left(\frac{m}{x}\right)^x\left(\ln\frac{m}{x} - 1\right) = 0 \quad \Leftrightarrow \quad \left(\ln\frac{m}{x} - 1\right) = 0$$

$$\ln\frac{m}{x} = 1 \implies \frac{m}{x} = e$$

Kapacita je maximálna, ak sú takmer všetky čísla rovné 3 (zaokrúhlenie čísla e). Keďže $(m-3) \mod 3 = m \mod 3$, tak počet riadkov bude nasledovný:

- ak $m \mod 3 = 0$, tak $\frac{m-3}{3}$ riadkov po troch funkciách
- ak $m \mod 3 = 1$, tak $\left\lfloor\frac{m-3}{3}\right\rfloor - 1$ riadkov po troch a jeden so štyrmi funkciami
- ak $m \mod 3 = 2$, tak $\left\lfloor\frac{m-3}{3}\right\rfloor$ riadkov po troch a jeden s dvoma funkciami

Pomocou m funkcií teda vieme reprezentovať $\Theta\left((\frac{m-3}{3})^3\right) = \Theta(m^3)$ bitov. Na zakódovanie n bitov potrebujeme

$$\lceil 3.\sqrt[3]{n}\rceil + 3$$

funkcií.

Na záver treba ešte zabezpečiť usporiadanie funkcií A a B (aby sme vedeli, či daná cesta kóduje 0 alebo 1). K dispozícií je viacero možností, napríklad volanie funkcií A a B z S v správnom poradí alebo použitie prvej cesty na určenie funkcie kódujúcej 0 (resp. 1).

Príklad 3.1. Nech má vodoznak 512 bitov. Ak ho rozdelíme na 16 úsekov po 32 bitoch, potrebujeme zakódovať 16 reťazcov po $32 + \log 16 = 36$ bitoch. Každý úsek vyžaduje $\lceil 3.\sqrt[3]{36}\rceil + 3 = 13$ funkcií, spolu potrebujeme $16.13 = 208$ funkcií. □

Graf použitý na kódovanie úsekov vodoznaku je pomerne hustý, čo môže byť pre útočníka podozrivé. Preto je vhodné zvýšiť odolnosť vynechaním niektorých hrán aj za cenu zníženia počtu ciest. Podobne pre zvýšenie utajenosti môžeme funkcie A a B nahradiť viacerými dvojicami funkcií, prípadne jednou k-ticou.

3.3.3 Výber WM funkcií

Funkcie kódujúce vodoznak musia byť programom používané. Vytvoríme ich z výrazov nachádzajúcich sa v programe a následne nahradíme výrazy volaním týchto funkcií. Programátor s pomocou vkladača určí miesta programu (triedy, metódy, funkcie a procedúry, balíky, zdrojové súbory atď.) na vyhľadávanie výrazov, ktoré budú základom pre vytvorenie nových WM funkcií. Nevhodnými miestami sú časti kódu, ktoré sa vykonávajú príliš často, napr. vnútra cyklov alebo globálne funkcie používané v celom programe. Vhodnejšie sú napr. inicializačné funkcie alebo kód obhospodarúvajúci užívateľské rozhranie programu.

Aby bolo nahradenie výrazu funkciou opodstatnené, treba vybrať výrazy, ktoré sa v programe vyskytujú viackrát. Takáto možnosť je však asi len pre jednoduché výrazy, lebo zložitejšie už nahradili funkciami programátori. Jednoduchosť výrazov by mohla byť problém pri optimalizácií inlinovaním, avšak aj jednoduché výrazy sa dajú počítať zložito, napríklad rekurzívne. Ďalšiu zložitosť do funkcie dodá kód implementujúci volanie WM funkcií (hrany grafu kódujúceho úsek vodoznaku).

3.3.4 WM funkcie

WM funkcie sú stavebným kameňom navrhovanej techniky. V rámci WM funkcie treba implementovať:

- volanie ďalších WM funkcií (za účelom kódovania vodoznaku)
- výpočet návratovej hodnoty funkcie

Volanie ďalších WM funkcií

Primárnou úlohou WM funkcie, vzhľadom na vodoznak, je volanie iných WM funkcií. Funkcie, ktoré nejaká WM funkcia volá, sú buď pevne dané, alebo závisia od konkrétneho vodoznaku. Adresy pevne daných funkcií môžu byť súčasťou statického kódu, ostatné adresy musíme dynamicky počítať počas behu programu. To, ktorá funkcia bude volaná, môže teda závisieť od

- aktuálnej WM funkcie (jej adresy, resp. aktuálnej hodnoty PC (Program Counter) registra[6])
- predošlej funkcie, resp. predošlých funkcií (ich adresy sa nachádzajú v zásobníku)
- poradia volanej funkcie medzi všetkými volanými funkciami
- špeciálnych dát programu (určujúcich konkrétny vodoznak danej kópie programu) vložených po kompilácii

Jediným spôsobom, ako na procesoroch architektúry IA32 zistiť adresu aktuálne vykonávaného kódu, je použitie inštrukcie `call` (volanie procedúry)[7] a následné prečítanie adresy na vrchu zásobníka, konkrétne

[6]na architektúre IA32 nazývaný IP, resp. EIP (Instruction Pointer)
[7]tiež inštrukcie INT, INT3, INTO a SYSENTER, ktoré však používa iba operačný systém

$$\begin{aligned} index &\leftarrow hash(Parametre) \\ \text{address} &\leftarrow T[index] \ \mathbf{xor} \ Parametre \end{aligned}$$

Obr. 3.2: Výpočet adresy volanej funkcie

```
1:         call label1
2:  label1: pop eax
```

Obsahom registra `eax` bude adresa inštrukcie `pop eax` (inštrukcia `call` vloží na zásobník adresu nasledujúcej inštrukcie).

Podobne vieme určiť adresu, z ktorej bola aktuálna funkcia volaná. Štandardné funkcie začínajú inštrukciami

```
push ebp
mov ebp,esp
```

teda na adrese `[ebp+4]` sa nachádza adresa volania aktuálnej funkcie (v skutočnosti adresa inštrukcie nasledujúcej inštrukciu `call`). Na adrese `[ebp]` sa nachádza predošlá hodnota registra `ebp`, pomocou ktorého sa dostaneme k adrese volania predošlej funkcie. Týmto spôsobom vieme zistiť miesta volaní všetkých predošlých funkcií.

Výpočet adresy nasledujúcej WM funkcie funguje podobne, ako pri technike DPB (2.2.6). Z parametrov ako aktuálna adresa v kóde, adresy predošlých funkcií, prípadne index volanej funkcie, vypočítame pomocou perfektnej hašovacej funkcie index do tabuľky adries v dátovej sekcii programu. Na tomto mieste sa v tabuľke nachádza hľadaná adresa v zakryptovanej forme. Dekrypčným kľúčom sú už spomenuté vstupné parametre pre výpočet indexu do tabuľky. Takýto výpočet znemožní staticky určiť index do tabuľky a jej skutočný obsah. Schéma s použitím funkcie *xor* je na obrázku 3.2.

V DPB technike sa pre každý vodoznak konštruuje zvláštna perfektná hašovacia funkcia. V prípade kódovania vodoznaku podľa princípu z 3.3.2 potrebujeme pre daný program jednu hašovaciu funkciu pre všetky vodoznaky (odtlačky), lebo parametre výpočtu adries volaných funkcií nezávisia od konkrétneho odtlačku.

Ak sú volania viacerých WM funkcií implementované cyklom (obr. 3.3), tak poradie volanej funkcie je parametrom pri výpočte jej adresy. Cyklus volania WM funkcií sa ukončí pri rovnosti vypočítanej adresy s nejakou konštantou. Poradie môže určovať aj špeciálny výpočet, keď funkcia obsahuje samostatný kód pre výpočet adresy každej volanej funkcie (obr. 3.4).

Na obrázku 3.5 je schématický príklad výpočtu adresy jednej funkcie na architektúre IA32. V praxi treba samozrejme jeho časti riešiť iným spôsobom.

WM-Function(parameters)

```
1: NasledujucaFunkcia ← Vypočítaj adresu nasledujúcej funkcie
2: while NasledujucaFunkcia je WM funkcia do
3:    Zavolaj funkciu NasledujucaFunkcia
4:    NasledujucaFunkcia ← Vypočítaj adresu nasledujúcej funkcie
5: end while
```

Obr. 3.3: Volanie WM funkcií v cykle

WM-Function(parameters)

```
1: NasledujucaFunkcia ← Vypočítaj adresu nasledujúcej funkcie
2: Zavolaj funkciu NasledujucaFunkcia
3: NasledujucaFunkcia ←Vypočítaj adresu nasledujúcej funkcie
4: Zavolaj funkciu NasledujucaFunkcia
5: NasledujucaFunkcia ←Vypočítaj adresu nasledujúcej funkcie
6: Zavolaj funkciu NasledujucaFunkcia
7: ...
```

Obr. 3.4: Volanie WM funkcií v rade

Pre zvýšenie odolnosti môžeme každú WM funkciu implementovať trochu inak, každá môže používať vlastnú hašovaciu funkciu (a hašovaciu tabuľku) a vlastný šifrovací algoritmus.

Maximálna veľkosť jednej hašovacej tabuľky závisí od počtu WM funkcií kódujúcich príslušnú časť vodoznaku. Teoretické maximum je $O(m.(m+k))$, lebo každá WM funkcia môže byť zavolaná z m WM funkcií alebo z k miest programu a vykonať m volaní WM funkcií (m je počet WM funkcií kódujúcich príslušnú časť vodoznaku, k je počet volaní WM funkcie z kódu programu). V závislosti od konkrétneho spôsobu kódovania úseku vodoznaku grafom sa toto maximum výrazne zmenšuje, napríklad ak sú kódujúce grafy stromy, tak priemerná veľkosť tabuľky je $\frac{2m-1}{m} + \frac{\sum k}{m}$.

Ak je skutočná veľkosť hašovacej tabuľky známa až po vložení vodoznaku, tak je rozumné nerezervovať miesto pre ňu pred vkladaním, lebo sa aj tak nevyužije efektívne. Namiesto toho budeme vkladať do dátovej sekcie skutočné hašovacie tabuľky. Zmiešame ich s dátami samotného programu – relokujeme niektoré z dát programu, čo vyžaduje uchovávať pri linkovaní potrebné informácie. Hoci sa veľkosť hašovacej tabuľky mení v závislosti od konkrétneho vodoznaku, suma veľkostí všetkých hašovacích tabuliek a teda aj veľkosť statickej dátovej sekcie ostáva tá istá.

```
; edi <- poradie nasledujúcej funkcie v rámci všetkých nasledovníkom
;         aktuálnej funkcie

; eax <- aktuálna adresa

 1:          call label1
 2:  label1: pop eax

; ebx <- adresa volania aktuálnej funkcie

 3:          mov ebx,[ebp+4]

; ecx <- adresa zašifrovanej adresy nasledujúcej funkcie

 4:          shl eax,7
 5:          shr ebx,5
 6:          mov ecx,eax
 7:          and ecx,ebx

; edx <- zašifrovaná adresa nasledujúcej funkcie

 8:          mov edx,[ecx]

; eax <- dešifrovací kľúč

 9:          or eax,ebx

; edx <- adresa nasledujúcej funkcie

10:          xor edx,eax
11:          call edx
```

Obr. 3.5: Výpočet adresy volanej WM funkcie

```
WM-DOUBLE(x)
 1: Next_WMF ← Výpočet adresy nasledujúcej WM funkcie
 2: case Next_WMF of
 3:   WM-ADD: return CALLFUNCTION(Next_WMF, array(x, x))
 4:   WM-TIMESTHREEMINUSTWO: return CALLFUNCTION(Next_WMF, array(x))−
      x + 2
 5:   WM-SQUARE: return CALLFUNCTION(Next_WMF, array(x)) − (x − 2) ∗ x
 6:   WM-LEFTSHIFT: return CALLFUNCTION(Next_WMF, array(x, 1))
 7:   WM-NOTHING: begin
 8:     if x = 1 then
 9:       return 2
10:     else
11:       return CALLFUNCTION(Next_WMF, array(x − 1)) + 2
12:     end if
13:   end
14: end case
```

Obr. 3.6: Funkcia WM-DOUBLE v. 1

Výpočet návratovej hodnoty funkcie

Výpočet návratovej hodnoty funkcie musí využívať WM funkcie volané za účelom kódovania vodoznaku, aby sa zabránilo odstráneniu týchto volaní vďaka odhaleniu ich zbytočnosti analýzou toku dát.

Kódovanie vodoznaku opísané v časti 3.3.2 používa WM funkcie, ktoré volajú najviac jednu vodoznakom danú WM funkciu (funkcie volajúce pomocné funkcie A a B) alebo pevné dané WM funkcie (funkcie tvoriace cesty pre kódovanie bitov vodoznaku).

Vo všeobecnosti, nech WM funkcia volá naraz najviac jednu z m WM funkcií. Do funkcie vložíme m rôznych výrazov počítajúcich jej hodnotu, každý z nich bude využívať inú WM funkciu.

Príklad 3.2. Kód na obrázku 3.6[8] ilustruje relevantnú časť jednoduchej WM funkcie WM-DOUBLE počítajúcej dvojnásobok vstupného parametra.

Ak by funkcia WM-DOUBLE bola posledná v reťazci funkcií kódujúcich vodoznak, tak musí sama vypočítať svoju hodnotu. Na riadkoch 8 až 11 je ukážka rekurzívnej implementácie výrazu $2*x$, ktorá chráni funkciu pred inlinovaním. Vyžaduje to však úpravu výpočtu

[8] parametrami funkcie CALLFUNCTION sú adresa volanej funkcie a pole jej parametrov

WM-DOUBLE(x)

1: $Next_WMF \leftarrow$ Výpočet adresy nasledujúcej WM funkcie
```
// Next_WMF = WM_TimesThreeMinusTwo
```
2: $Vysledok \leftarrow$ CALLFUNCTION($Next_WMF, array(x)$)

3: $Next_WMF \leftarrow$ Výpočet adresy nasledujúcej WM funkcie
```
// Next_WMF = WM_Add
```
4: $Vysledok \leftarrow$ CALLFUNCTION($Next_WMF, array(Vysledok, x + 2)$)

5: $Next_WMF \leftarrow$ Výpočet adresy nasledujúcej WM funkcie
```
// Next_WMF = WM_LeftShift
```
6: **return** CALLFUNCTION($Next_WMF, array(Vysledok, -1)$)

Obr. 3.7: Funkcia WM-DOUBLE v. 2

adresy nasledujúcej WM funkcie tak, aby po zavolaní funkcie WM-DOUBLE sebou samou bola nasledujúca funkcia WM-NOTHING, teda lokálny výpočet návratovej hodnoty.

Na obrázku 3.7 je kód druhej verzie funkcie WM-DOUBLE, volajúcej pevne dané funkcie (hoci ich adresy sú počítané až počas behu programu). □

Použitie aktuálnej adresy vo výpočte návratovej hodnoty môže slúžiť na ochranu proti relokovaniu WM funkcie. Kód takéhoto výpočtu však môže byť vygenerovaný až pri linkovaní programu, keď už poznáme absolútne adresy. Takéto možnosti však neposkytuje každý kompilátor resp. linker, vyžadovalo by to pravdepodobne jeho úpravu, čo nie je vždy možné.

Situácia je zložitejšia v prípade kódovania vodoznaku grafmi, teda keď WM funkcia volá viac než jednu inú vodoznakom určenú WM funkciu. WM funkcia nemôže prakticky využiť volané WM funkcie pre výpočet svojej hodnoty, lebo pre každú kombináciu volaných WM funkcií, ktorých je exponenciálne veľa vzhľadom na počet funkcií, potrebuje jeden výraz. Nutnosť volania WM funkcií sa môžeme snažiť zabezpečiť nasledovne:

- volané WM funkcie použijeme pre výpočet návratovej hodnoty, avšak vplyv každej volanej funkcie neutralizujeme, napríklad $Vysledok =$ WM-DOUBLE($Vysledok$)/2. Takéto správanie (priradzovanie rovnakej hodnoty do premennej) je však odhaliteľné.
- upravíme program a do WM funkcií vložíme príslušný kód tak, aby vykonanie WM funkcií v správnej chvíli bolo nevyhnutné pre správny beh programu

3.4 Rozpoznávanie vodoznaku

Rozpoznávanie vodoznaku prebieha v štyroch fázach:

1. Konštrukcia grafu volania funkcií programu
2. Rozpoznanie WM funkcií vďaka pomocnej informácii
3. Konštrukcia úsekov vodoznaku z grafov volania WM funkcií
4. Konštrukcia vodoznaku z úsekov pomocou ID úsekov

3.4.1 Konštrukcia grafu volania funkcií programu

Prvá fáza je nezávislá na použitej technike. Graf volania funkcií je podmnožinou grafu toku riadenia programu, ktorý vieme zostrojiť postupom opísaným v časti 3.1.1. Zkonštruujeme ho ztotožnením vrcholov, ktoré sú súčasťou jednej funkcie, do jedného vrcholu, nahradením viacnásobných hrán jednoduchými hranami a odstránením slučiek. Ak implementácia potrebuje sledy v grafe volania funkcií, je potrebné ich vytvoriť transformáciou sledov grafu toku riadenia (podobne ako vytvorenie grafu volania funkcií z grafu toku riadenia).

3.4.2 Rozpoznanie WM funkcií

Cieľom je rozpoznať v grafe volania funkcií podgrafy kódujúce úseky vodoznaku. Venujme sa najprv hľadaniu jedného takéhoto podgrafu, resp. hľadaniu jeho funkcií. Jeho rozpoznávanie môže byť založené na:

- rozpoznaní jednej funkcie a následnom rozpoznaní zvyšku podgrafu
- rozpoznaní celého podgrafu naraz

Rozpoznanie jednej funkcie

WM funkcie vyzerajú ako normálne funkcie programu, preto problém ich rozpoznania je ekvivalentný s problémom nájdenia ľubovoľnej inej funkcie programu. Rozpoznávanie využíva kľúč vodoznaku, ktorý je výstupom vkladania vodoznaku do programu. Hľadanie jednej funkcie nezávisí od konkrétneho vodoznaku, lebo WM funkcie sú pre každý vodoznak rovnaké (kód programu sa nemení v závislosti od konkrétneho vodoznaku).

Nezávisle od grafu volania funkcií môžeme hľadať funkciu v samotnom kóde programu na základe znalosti jej veľkosti, kódu alebo adresy umiestnenia. V rámci grafu volania

funkcií máme viacero možností, avšak generovanie kľúča vyžaduje znalosť rozpoznateľného grafu, t. j. nie skutočného grafu, ale grafu, ktorý je schopný vytvoriť rozpoznávač. Konštrukcia tohto grafu vyžaduje spoluprácu kompilátora (pozná funkcie a volania v rámci programu), vkladača vodoznaku (pozná volania medzi WM funkciami vodoznaku) a rozpoznávača (pozná, resp. dokáže vytvoriť rozpoznateľný graf).

Kľúčom pre hľadanie funkcie v grafe volaní môže byť

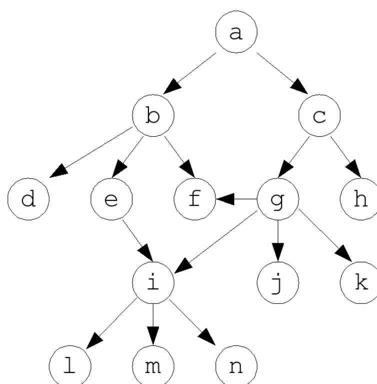
- index funkcie v rámci očíslovania vrcholov grafu pre-order alebo post-order prechodom
- súradnice vrchola – hĺbka funkcie v grafe (riadok) a poradie z ľava (stĺpec)
- cesta k vrcholu – nejde tu o cestu v zmysle teórie grafov, ale o postupnosť čísel, ktoré navigujú k hľadanej funkcii. Jedna položka cesty hovorí o synovi nejakého vrchola. Jej hodnota n môže znamenať:
  - index syna v rámci všetkých synov – synom je tá funkcia, ktorá je volaná volaná ako n-tá v poradí. Môžu však existovať vrcholy, ktorých synovia nie sú vždy volaní v rovnakom poradí, napr. ak je adresa volanej funkcie (syn) dynamicky určená na základe hodnoty parametra funkcie (otec) a tento parameter závisí od konkrétnych vstupov programu.
  - počet vlastných synov – synom je tá funkcia, ktorá má n synov. Problematická je situácia, keď dvaja (alebo viacerí) synovia majú rovnaký počet synov. V tom prípade treba otestovať obe vetvy, t. j. pokračovať s vnukmi a ich synmi. Aj takto však môžu existujú dve (alebo viac) ciest.
  - počet parametrov – synom je funkcia, ktorá má n parametrov. Ten nemusí byť možné vždy určiť priamo. Na IA32 architektúre môžeme využiť inštrukciu `ret` x, ktorá vráti riadenie za miesto volania funkcie a zo zásobníka odstráni x bajtov – parametre funkcie. Podobne ako v predošlom prípade, v grafe môže existovať viacero rovnakých ciest, tentoraz je to dokonca pravdepodobnejšie.

Nejednoznačnosť však v skutočnosti nenastane, lebo k funkciám existuje z koreňa viacero rôznych ciest. Vkladač dokáže odhaliť ich nejednoznačnosť a na generovanie kľúča použije iba jednoznačné cesty.

Zlepšením uvedeného postupu je zaviesť redundanciu použitím kombinácie niektorých z uvedených spôsobov. Teda jednu alebo viacero ciest z vrchola určíme viacerými spôsobmi, napr. prvú cestu pomocou indexu syna a počtu synov a druhú cestu pomocou počtu synov a počtu parametrov.

Príklad 3.3. Na obrázku 3.8 je graf volania funkcií s koreňom `a`. Nech vrchol `i` je koreňom hľadaného podgrafu. Z koreňa `a` existujú do vrchola `i` dve cesty, ktoré môžeme zakódovať nasledovne:

1. cesta `a,b,e,i` – index syna: 1, 2, 1; počet synov: 3, 1, 3
2. cesta `a,c,g,i` – index syna: 2, 1, 2; počet synov: 2, 4, 3



Obr. 3.8: Graf volania funkcií

□

Podgraf kódujúci vodoznak je acyklický, preto rozpoznanie podgrafu z jednej funkcie vyžaduje, aby buď táto bola koreňom podgrafu, alebo aby kľúč obsahoval potrebné dodatočné informácie, ako napríklad:

- umiestnenie funkcie v rámci podgrafu – napríklad v prípade kódovania vodoznaku podľa 3.3.2 môže byť nájdená funkcia v prvom riadku funkcií a pomocná informácia hovorí o jej umiestnení v riadku. Ak sú všetky funkcie prvého riadku takto identifikované, nie je nutné použiť koreň S
- navigáciu ku koreňu podgrafu

Znalosť koreňa podgrafu umožňuje nájsť celý podgraf, lebo z koreňa existuje cesta do každého vrchola. Avšak nájdený podgraf môže byť nadgrafom skutočne hľadaného podgrafu, lebo obsahuje volania funkcií run-time podpory. Rozpoznávač však tieto funkcie pozná (ako súčasť kľúča alebo vstavanú informáciu) a je schopný podgraf od nich očistiť.

Rozpoznanie podgrafu

Kľúčom k rozpoznaniu podgrafu v grafe je napríklad opis topológie. Rozpoznávanie prebieha prechodom cez všetky vrcholy, pri ktorom pre každý vrchol grafu testujeme, či môže byť koreňom hľadaného podgrafu.

Niektoré (prípadne všetky alebo žiadne) hrany podgrafu však môžu byť závislé na konkrétnom vodoznaku, čo znamená, že kľúč opisuje iba podgraf hľadaného podgrafu. Túto metódu teda môžeme použiť len vtedy, ak je hľadaný podgraf dostačne nezávislý od konkrétneho vodoznaku nato, aby opis jeho topológie bol jednoznačný, t. j. aby sme pri prechode grafu programu nenašli viac ako jeden vrchol, ktorý by mohol byť koreňom podgrafu. O tejto jednoznačnosti, a teda aj o použití takéhoto kľúča, vieme rozhodnúť pri vkladaní vodoznaku (podobne ako jednoznačnosť ciest určujúcich funkcie). Úplná nezávislosť nemusí byť nutná, ale ani postačujúca pre nájdenie jediného podgrafu.

V prípade závislosti podgrafu od konkrétneho vodoznaku je možné zabezpečiť jednoznačnosť pridaním závislých hrán do opisu topológie podgrafu – kľúč bude závislý na konkrétnom vodoznaku. Stratí sa tým nezávislosť na vodoznaku, čo v prípade použitia rýdzeho vodoznaku (1.1) nie je problém.

Rozpoznanie ostatných podgrafov môže prebiehať nezávisle od ostatných (použitím nejakého zo spomenutých spôsobov) alebo bude závisieť na prvom rozpoznanom grafe, resp. niekoľkých nezávisle rozpoznaných podgrafoch. Ide v podstate o vybudovanie grafu nad podgrafmi. Týmto grafom môže byť

- lineárny reťazec
- strom
- všeobecný súvislý zakorenený graf

Hrany tohto grafu môžu v prípade novej techniky tvoriť volania medzi pomocnými funkciami S – koreňmi jednotlivých podgrafov. Koreň volaný koreňom vráti riadenie naspäť, aby sa nevolalo zbytočne veľa funkcií, čo by bolo neefektívne a podozrivé.

Výhody a nevýhody oboch prístupov zhŕňa nasledovný prehľad:

- Nezávislé podgrafy
  - vyžadujú väčší kľúč – každý podgraf je identifikovaný samostatne
  - vyžaduje ID úsekov vodoznaku – úseky treba vedieť zoradiť
  - vyššia odolnosť – napadnutie jedného podgrafu neohrozuje ostatné

- Graf podgrafov
  - menší kľúč – stačí identifikovať koreň, rozpoznanie ostatných podgrafov vyžaduje pre každý najviac konštantné množstvo informácie
  - nevyžaduje explicitné usporiadanie grafov (úsekov) – ID úseku je ID podgrafu v rámci usporiadania podgrafov v grafe, napr. pre-order alebo post-order prechodom
  - nižšia odolnosť – odhalenie nejakého podgrafu môže mať za následok odhalenie ďalších podgrafov, keďže sú navzájom prepojené

  Výstupom tejto fáza je zoznam podgrafov (usporiadaný alebo neusporiadaný) a ku každému z nich prislúchajúce sledy.

3.4.3 Konštrukcia vodoznaku

Konštrukcia úseku vodoznaku z podgrafu je priamočiara, závisí len od spôsobu kódovania informácií grafom. V prípade kódovania informácií z časti 3.3.2 existuje ku každému podgrafu jeden sled začínajúci v jeho vrchole. Sled kóduje informácie svoju podpostupnosťou obsahujúcou vrcholy A a B, t.j. bity 0 a 1.

Nasleduje spojenie úsekov do výsledného reťazca, ktoré závisí od spôsobu ich usporiadania (pomocou ID alebo podľa usporiadania podgrafov). Ak bol použitý nejaký samoopravný kód, tak poslednou fázou je dekódovanie získaného reťazca.

3.5 Odolnosť voči útokom

3.5.1 Odstránenie vodoznaku

Útočník sa snaží nájsť tie časti programu, ktoré kódujú vodoznak, s cieľom ich odstránenia. Navrhovaná technika sa snaží riešiť nedostatky DPB techniky hlavne v oblasti odhalenia častí programu kódujúcich vodoznak (t. j. WM funkcie).

Štatistická analýza

Jedným z cieľom pri návrhu novej techniky bolo zvýšiť odolnosť voči útoku založenom na štatistickej analýze. Preto za stavebné kamene vodoznaku boli vybraté funkcie a volania

medzi nimi, ktoré sú nevyhnutnou súčasťou každého štruktúrovaného programu. Ich atribúty však nesmú vybočovať zo štandardných hraníc a tiež by nemali ovplyvniť priemerné hodnoty týchto atribútov pre celý program. Medzi uvažované atribúty funkcií patria

- počet bajtov kódu funkcie
- počet volaní funkcie
- počet funkcií volaných funkciou

Konkrétne hodnoty týchto atribútov závisia od implementácie. Požiadavky, ktoré navrhovaná technika kladie na WM funkcie, udržia hodnoty atribútov v požadovaných medziach. V porovnaní s DPB, ktorej vetviaca funkcia bola volaná z neúmerne veľkého počtu miest v programe, je teda odolnosť novej techniky vyššia.

Podobne sa treba vyvarovať použitiu netypických konštrukcií, resp. kódu, ktorý nie je vygenerovateľný kompilátorom. V prípade DPB sa jedná

- o zmenu návratovej adresy funkcie
- o cykly dĺžky viac než 1 v grafe volania funkcií

Oba tieto nedostatky nová technika odstraňuje.

Diferenčný útok

Kópie programu s rôznym vodoznakom sa nelíšia v kóde, iba v dátach. Ich porovnávaním útočník odhalí rôzniace sa časti dát. Dynamická analýza programu môže odhaliť miesta v kóde (WM funkcie), ktoré referencujú rôzniace sa dáta. Proti tomuto útoku sa dá brániť dvoma spôsobmi:

1. Pridanie ďalších dát, ktoré budú rôzne pre každý vodoznak – porovnávané súbory s rôznymi vodoznakmi sa budú líšiť vo veľkom množstve miest, čo zvýši čas potrebný na analýzu všetkých miest, ktoré referencujú rôzniace sa dáta. Okrem pridania nových dát však treba zabezpečiť aj miesta, ktoré budú tieto dáta referencovať tak, aby sa tieto referencie nedali odstrániť optimalizáciou.

2. Pridanie referencií na vodoznakové dáta – zvýši sa tým množstvo miest v kóde, ktoré je treba analyzovať. Podobne ako v predošlom prípade to vyžaduje upraviť niektoré miesta v kóde programu tak, aby využívali dané dáta a aby sa referencie nedali odstrániť optimalizáciou.

Vkladanie nového a úprava existujúceho kódu tak, aby referencoval potrebné dáta a alokácia miesta pre nové dáta sa vykonávajú nad zdrojovým kódom programu spolu s vkladaním WM funkcií.

Optimalizácia odstránením mŕtveho kódu

Detekcia mŕtveho kódu je síce nerozhodnuteľný problém, avšak v praxi často (čiastočne) riešený. WM funkcie sú však používané (staticky aj dynamicky) programom, a preto nie sú kandidátmi na odstránenie ako mŕtvy kód.

Odstránenie vodoznaku

Hlavný dôraz novej techniky je kladený na zabránenie detekcie WM funkcií. Nedá sa jej však zaručene predísť, a preto je technika navrhnutá tak, aby aj odstráneniu odhalených WM funkcií a ich vzájomných volaní čo najviac bránila.

WM funkcie nie je možné jednoducho odstrániť, pretože počítajú pre program dôležité výpočty. Útočník musí nahradiť WM funkciu inou funkciou (t. j. upraviť WM funkciu tak, aby výpočet hodnoty nevyžadoval volanie ďalších WM funkcií). Vyžaduje to však znalosť výrazu, ktorý funkcia počíta. K hľadaniu výrazu sa dá pristúpiť dvoma spôsobmi:

1. Sledovanie vzťahu medzi vstupnými a výstupnými parametrami – určenie výrazu na základe týchto informácií nie je vo všeobecnosti možné

2. Extrakcia výrazu z kódu WM funkcií – vyžaduje analyzovať tok dát v grafe toku riadenia programu. Získaný výraz je potrebné optimalizovať. Počítanie výrazu WM funkciami je však implementované tak, aby v čo najväčšej miere bránilo optimalizácií (používanie rekurzie, miešanie celých a reálnych čísel atď.)

Nahradenie volania WM funkcií vo vnútri WM funkcie môže znamenať zväčšenie jej kódu. Takže útočník musí vyvinúť ďalšie úsilie na úpravu programu, lebo jednoduché nahradenie funkcie má za následok zmenu adries ďalšieho kódu. Neošetrenie tejto situácie vyústi v nefunkčnosť programu.

Ochranou proti odstráneniu vzájomných volaní WM funkcií je použitie návratových hodnôt volaných WM funkcií vo výpočte vlastnej návratovej hodnoty (pozri *Výpočet návratovej hodnoty funkcie* v časti 3.3.4).

3.5.2 Skreslenie vodoznaku

Ochranou voči statickým útokom je používanie dynamicky počítaných adries volaných funkcií. Ľubovoľná zmena programu, ktorá ovplyvní adresy týchto funkcií spôsobí nefunkčnosť programu.

Úspešnosť dynamického útoku závisí na úspešnosti konštrukcie grafu toku riadenia programu a na vykonanej transformácii.

Vzhľadom na samotný princíp ukladania vodoznaku je mnoho útokov bezpredmetných, pretože nemenia tok riadenia programu (čoho súčasťou sú volania WM funkcií). Tok riadenia je zároveň tým, čo robí každý program originálnym (operačná sémantika). Na druhej strane však útočník potrebuje zachovať len vstupno-výstupnú sémantiku. Môže sa teda snažiť aplikovať (na celý program) tieto transformácie s cieľom zmeniť graf volania funkcií:

- zmena poradia volania funkcií – je ju možné vykonať len v prípade, že na poradí volania daných funkcií nezáleží, čomu sa snažíme pri vkladaní vodoznaku zabrániť
- inlinovanie funkcií – znamená nahradenie volania funkcie jej telom (jedna z optimalizácií vykonávaných kompilátormi). Čím je uvažovaná funkcia väčšia, tým je inlinovanie náročnejšie, lebo znamená úpravu väčšej časti programu, čo môže znamenať, že inlinovanie nebude úspešné, lebo by zasiahlo časti programu, ktoré nie sú korektne disassemblované. Účinnou obranou pre WM funkcie je ich rekurzívna implementácia, inlinovanie takýchto funkcií nezmení graf volania funkcií. Inlinovanie ostatných funkcií programu odstráni hrany, príp. vrcholy grafu volania funkcií a čo môže sťažiť až znemožniť identifikáciu WM funkcií pri rozpoznávaní vodoznaku.
- outlinovanie funkcií – nahradenie časti kódu volaním funkcie, ktorá obsahuje tento kód. Úspešnosť útoku závisí na možnosti pridávať do programu nový kód (nové funkcie). V úspešný útok by znamenal pridanie nových vrcholov a hrán do grafu volania funkcií, čo, podobne ako pri inlinovaní, môže znemožniť nájdenie WM funkcií pri rozpoznávaní.

3.5.3 Pridanie ďalšieho vodoznaku

Podobne ako pri skresľovacom útoku, pridanie kódu kódujúceho nový vodoznak, vyžaduje konštrukciu grafu toku riadenia. Hlavným cieľom navrhovanej techniky však nie je dôkaz autorstva softvéru, ale identifikácia zákazníka porušujúceho licenčnú zmluvu, a preto pridanie iného vodoznaku nie je problémom, ak pôvodný vodoznak ostane zachovaný.

3.6 Ďalšie kritériá kvality

3.6.1 Kapacita

Vloženie vodoznaku znamená rozšírenie programu o príslušný počet WM funkcií a dáta kódujúce volania WM funkcií. Veľkosť pridaného kódu je sumou veľkostí kódov jednotlivých WM funkcií, teda

$$\text{veľkosť kódu} = \sum_{i=1}^{\#\text{WM funkcií}} \text{veľkosť kódu } i\text{-tej WM funkcie}$$

Množstvo dát kódujúcich volania WM funkcií závisí od implementácie. V prípade kódovania úsekov vodoznaku podľa 3.3.2 je to

$$\text{veľkosť dát} = 2 \sum_{i=1}^{\#\text{úsekov}} \prod_{j=1}^{\#\text{vrstiev}} \#r_{i,j}$$

Celková kapacita je teda

$$Kapacita = \frac{\#\text{ bitov vodoznaku}}{\text{veľkosť kódu} + \text{veľkosť dát}} =$$

$$= \frac{\#\text{ bitov vodoznaku}}{\sum_{i=1}^{\#\text{WM funkcií}} \text{veľkosť kódu } i\text{-tej WM funkcie} + 2\sum_{i=1}^{\#\text{úsekov}} \prod_{j=1}^{\#\text{vrstiev}} \#r_{i,j}} \text{bitov na bajt}$$

Príklad 3.4. Ak pre 512 bitový vodoznak (16 úsekov po 32 bitov) potrebujeme 208 funkcií (príklad 3.1), tak pri 150 bajtoch na jednu funkciu[9] potrebujeme 208.150 = 31200 bajtov a pre dáta 2.16.36.4 bajtov = 4608 bajtov. Spolu 35808 bajtov, čo predstavuje kapacitu 512/35808 = 0,014 bitu na bajt, t. j. 1 bit na 70 bajtov. □

3.6.2 Efektivita

Označenie programu vodoznakom zväčší jeho veľkosť iba minimálne.

Veľkosť zvýšenia časových nárokov označeného programu závisí do veľkej miery od používateľa, ktorý vyberá miesta v programe, v ktorých môžu byť výrazy nahradené volaniami WM funkcií. Pri zavolaní koreňovej WM funkcie dochádza k

$$2 \prod_{j=1}^{\#\text{vrstiev}} \#r_{i,j}$$

[9] hrubý odhad priemeru pre architektúru IA32, podľa [16] je priemerná veľkosť inštrukcie asi 3 bajty a podľa [17] je priemerný počet inštrukcií vo vykonaných funkciách asi 50 (WM funkcie by mali „vyzerať priemerne“)

ďalším volaniam WM funkcií. K nim treba pripočítať volania nekoreňových WM funkcií. Ak volajú ďalšie WM funkcie, tak je počet volaní rádovo taký, ako počet volaní pri zavolaní koreňovej WM funkcie.

Príklad 3.5. Ak má program veľkosť 1MB, tak podľa predošlého príkladu ho 512 bitový vodoznak zväčší o 35808 bajtov, t. j. $3,41\%$. Počet volaní v rámci 32 bitového úseku kódovaného 13 funkciami je $2.36 = 72$. Ak je WM funkcia volaná v priemere k krát[10], znamená to $16.72.k + 16.(72 + 24 + 8 + 2).k = 1152.k + 1696.k = 2848.k$, resp. $16.72.k + 16.12.k = 1152.k + 162.k = 1344.k$ volaní. □

3.7 Porovnanie techník

Naša technika je navrhnutá s cieľom odstrániť niektoré z nedostatkov DPB techniky, a to hlavne v oblasti utajenosti a odolnosti.

3.7.1 Utajenosť

Technika používa pre uloženie vodoznaku funkcie, ktoré sú súčasťou každého zmysluplného programu. Sú vložené do zdrojového kódu programu, ich štatistické vlastnosti (veľkosť, počet volaní atď.) nevybočujú z normálu ako v prípade DPB techniky a ich kód je vygenerovateľný kompilátorom.

3.7.2 Odolnosť

Základným mechanizmom pre zvýšenie odolnosti voči útokom odstraňujúcim vodoznak je zlepšenie utajenosti. Technika je navrhnutá tak, aby vloženie vodoznaku len minimálne zmenilo štatistické vlastnosti programu. Preto je lokalizácia vodoznaku v porovnaní s DPB technikou náročnejšia.

Ďalším prostriedkom k odhaleniu vodoznaku v programe je diferenčný útok – porovnávanie viacerých kópií programu s rôznymi odtlačkami. Naša technika spôsobuje rozdielnosť dátových sekcií, samotný program ostáva nezmenený. V časti 3.5.1 uvádzame niektoré návrhy pre zvýšenie odolnosti pred týmto typom útoku.

Úspešná lokalizácia vodoznakových štruktúr neumožňuje ich jednoduché odstránenie vďaka ich vzájomnej závislosti. Konkrétne detaily závisia od implementácie. V porovnaní s DPB technikou je odstránenie vodoznaku náročnejšie.

[10]počet volaní WM funkcie závisí od konkrétneho programu a jeho vstupov

vlastnosť	z nej plynúca výhoda
lepšia utajenosť	znižuje šancu lokalizácie vodoznakových štruktúr
decentralizácia	odstraňuje bod zlyhania
heterogénnosť	znemožňuje plne automatizovaný útok, zvyšuje náklady na útok
zložitosť	zvyšuje náklady na útok

Tabuľka 3.1: Výhody navrhnutej techniky v porovnaní s DPB

Dôležitým rozdielom oproti DPB technike je decentralizácia vodoznaku, ktorá bráni odstráneniu celého vodoznaku po odhalení jednej, resp. niekoľkých jeho častí.

Väčšia zložitosť našej techniky oproti iným technikám taktiež pomáha odolnosti, lebo zvyšuje náročnosť tvorby pomocných nástrojov pre útočníka a taktiež časové nároky potenciálneho útoku.

Odolnosť voči skresľujúcim útokom je založená na dedičstve DPB techniky, ktorá patrí medzi najodolnejšie v tejto kategórii útokov. V časti 3.5.2 sa zaoberáme možnosťami skresľujúceho útoku, založeného na skonštrouvanom grafe toku riadenia (voči DPB technike takýto útok nie je potrebný, lebo znalosť grafu toku riadenia vedie k odstráneniu vodoznaku). Identifikovali sme čiastočnú zraniteľnosť in/out-linovaním funkcií, ktoré vedie k zmene grafu volania funkcií. Konkrétne možnosti útoku a obrana voči nim závisia na implementácii.

Pridanie vodoznaku, ktoré zmení adresy funkcií v programe, spôsobí jeho nefunkčnosť. Prítomnosť iného vodoznaku nie je problémom, keďže naša technika je určená pre odtlačky a nie dokazovanie pred súdom.

3.7.3 Kapacita

Nižšia kapacita (rádovo desiatky bajtov na jeden bit vodoznaku) je spôsobená použitím funkcií na kódovanie vodoznaku a je daňou za zvýšenú odolnosť.

3.7.4 Efektivita

Označený program je väčší (počet bajtov) oproti originálu vďaka pridaniu WM funkcií. Kvôli kódovaniu vodoznaku funkciami je zväčšenie väčšie v porovnaní s DPB technikou. Niektoré techniky dokonca nemenia veľkosť programu vôbec, napr. (najmä statické) techniky založené na preusporiadaní a premenovaní (inštrukcií, základných blokov atď.).

Ako už bolo napísané v časti 3.6.2, zvýšenie časových nárokov programu po označení závisí do veľkej mieri od miest, do ktorých bol vodoznak vložený (miesta volania WM

	alokácia registrov	**GTW**	**SWuS**	**DPB**	**náš návrh**
počet bajtov WM štruktúr na jeden bit vodoznaku	desiatky až stovky	desiatky	desiatky	bajty	desiatky
zväčšenie veľkosti[11]	žiadne	výrazne	mierne	málo	mierne
zvýšenie časových nárokov	žiadne	okolo 20%	minimálne	minimálne	závisí od vloženia

Tabuľka 3.2: Porovnanie kapacity a efektivity niektorých techník

funkcií). Podobne ako v prípade veľkosti označeného programu, existujú techniky, ktoré nemenia časové nároky programu.

3.7.5 Zhrnutie

Predstavená technika nie je technikou v zmysle definície 1.2, lebo nešpecifikuje všetky detaily implementácie. Umožňuje používateľovi podľa jeho potrieb zvoliť najvhodnejší prístup. Definuje základný rámec, opisuje možnosti, ktoré sú na výber, a hovorí o problémoch, ktoré je treba riešiť.

Technika spĺňa základné požiadavky kladené na watermarking techniky (1.2). Zverejnenie jej použitia pre nejaký program neohrozuje vložený vodoznak. Jeho rozpoznanie je podmienené znalosťou kľúča, ktorý musí ostať utajený. Proces rozpoznávania vodoznaku je verejný a automatizovaný, a preto odstraňuje pochybnosti o pravosti nájdeného vodoznaku. Požiadavka dokázateľnosti nie je na prvý pohľad splnená, keďže je v protiklade s utajenosťou (def. 1.5), jej riešenie však ponechávame na implementáciu, od ktorej náš návrh abstrahuje.

Výhody navrhovanej techniky spočívajú hlavne vo zvýšení odolnosti voči útokom. Hoci je čiastočne zraniteľná voči in/out-linovaniu funkcií, v porovnaní s inými technikami je odolná voči mnohým útokom:

- Odstránenie
  - štatistický útok
  - diferenčný
  - odstránenie mŕtveho kódu

[11] percentuálne zväčšenie závisí od veľkosti programu

- odstránenie po nájdení (čiastočne)

- Skreslenie
  - všetky statické útoky
  - dynamické útoky[12]
    - preusporiadanie (základných blokov) a premenovanie (registrov, inštrukcií)
    - obfuskácia

- Pridanie
  - pridanie vodoznaku, ktoré zmení adresy funkcií

Medzi nevýhody nášho návrhu patria

- nižšia kapacita
- vyššia náročnosť implementácie:
  - zložitejšie softvérové nástroje
  - vyžaduje manuálny zásah používateľa

3.8 Požiadavky

Zhrňme na záver požiadavky, ktoré kladie použitie navrhovanej techniky na vkladací a rozpoznávací softvér a na ich používateľov.

Požiadavky na používateľov

Najdôležitejšou požiadavkou kladenou na používateľov je určenie časti programu, v ktorých sa môžu vyhľadať výrazy pre nové WM funkcie. Táto činnosť sa vykonáva za pomoci vkladača.

Požiadavky na softvér

Podľa fázy vkladania delíme požiadavky na vkladací softvér na dve skupiny. Prvou fázou je vkladanie funkcií, ktoré vyžaduje:

[12] založené na konštrukcii grafu toku riadenia

1. znalosť gramatiky použitého programovacieho jazyka
2. spracovanie vstupu od používateľa
   - veľkosť vodoznaku
   - časti programu, v ktorých môžu byť výrazy nahradené WM funkciami
   - špecializujúce parametre, napr. výber spôsobu kódovania informácií grafmi, počet úsekov, na ktoré má byť vodoznak rozdelený atď.
3. nájdenie a vhodných výrazov a ich nahradenie volaniami WM funkcií
4. vytvorenie WM funkcií a ich vloženie do zdrojového kódu
5. vygenerovanie kľúča pre nájdenie WM funkcií pri rozpoznávaní

Fáza vkladania vodoznaku kladie nasledovné požiadavky:

1. spracovanie vstupu (vodoznak)
2. rozdelenie vodoznaku na úseky a ich zakódovanie do grafov
3. vygenerovanie pomocných dát na základe
   - dešifrovacích algoritmov vo WM funkciách
   - adries WM funkcií
   - grafov kódujúcich vodoznak
   - umiestnenia pomocných dát
4. vloženie pomocných dát do dátovej sekcie programu

Rozpoznávací softvér musí zvládnuť tieto operácie:

1. konštrukcia grafu volania funkcií
2. identifikácia WM podgrafov v grafe volania funkcií pomocou kľúča vodoznaku
3. dekódovanie WM podgrafov
4. spojenie úsekov do hľadaného reťazca a prípadne jeho dekódovanie

Každá z uvedených požiadaviek vyžaduje istú časovú investíciu. Podľa frekvencie vykonávania môžeme požadované činnosti usporiadať nasledovne (vzostupné poradie):

1. vývoj generického softvéru na vkladanie a rozpoznávanie vodoznakov, ktorý je nezávislý od konkrétnych hodnôt parametrov techniky (napr. kódovanie vodoznaku grafmi)

2. vývoj modulov implementujúcich zvolené algoritmy (ak ešte neboli implementované, alebo ak nie sú k dispozícii)

3. označenie konkrétneho programu (určenie časti programu, v ktorých sa môžu vyhľadať výrazy pre WM funkcie)

3.9 Príklad použitia

Nech firma PerfectSoft vyvíja program PerfectOffice a chce ho predávať a distribuovať pomocou Internetu. Zároveň sa snaží minimalizovať nelegálne používanie svojho produktu, a preto chce postihovať neoprávnených používateľov a tých, ktorí poskytli svoju kópiu tretej osobe.

Jedným z riešení tohto problému je použitie watermarking-u na vkladanie odtlačkov zákazníkov do programu a odmietnutie poskytovania podpory používateľom kópie programu, ktorá je predmetom neoprávneného nakladania.

Firma PerfectSoft si vyberie našu techniku pre vkladanie informácie o autorských právach a identifikačných čísel zákazníkov do svojho programu PerfectOffice. Na začiatku si zaobstará (vytvorí, použije slobodnú implementáciu alebo kúpi) generický balík pre vkladanie a rozpoznávanie vodoznakov. Podobne získa potrebné moduly pre konkrétne algoritmy.

Firma sa rozhodne, že vodoznak (odtlačok) bude obsahovať tieto položky:

- názov programu – `PerfectOffice`
- názov firmy – `PerfectSoft`
- rok vydania – `(c) 2007`
- číslo zákazníka – napr. 0x0012f2a3
- dátum predaja softvéru – napr. 7. 5. 2007

Následne určí miesta v zdrojovom kóde programu, v ktorých môže vkladač hľadať výrazy pre nahradenie WM funkciami. Potom spustí vkladač, ktorý vytvorí príslušné WM funkcie a vráti kľúč, ktorý bude uložený na bezpečné miesto.

Kupovanie softvéru bude fungovať nasledovne:

- zákazník sa zaregistruje na webovej stránke firmy
- zákazník sa rozhodne kúpiť softvér PerfectOffice a zaplatí príslušnú cenu
- spustí sa vkladač, ktorý vytvorí kópiu polooznačeného programu a vloží do nej odtlačok zákazníka
- takto označený program je sprístupnený zákazníkovi na stiahnutie
- zákazník si program stiahne, nainštaluje a následne používa (spolu s podporou firmy)

Ak pracovníci firmy zistia, že na niektorej *peer to peer* sieti sa šíri kópia ich programu PerfectOffice, tak si ju tiež stiahnu. Na stiahnutom programe spustia rozpoznávač spolu s kľúčom. V rozpoznanom vodoznaku nájdu číslo zákazníka a príslušnému zákazníkovi zrušia registráciu a poskytovanie podpory. Dôsledkom je, že všetci používatelia tej istej kópie programu stratia prístup k podpore firmy.

Záver

Watermarking spustiteľného kódu sa zaoberá vkladaním tajných správ, vodoznakov, do programov. Vložený vodoznak má za cieľ preukázať autorské práva tvorcov programu alebo vystopovať osobu, ktoré poskytla softvér na neoprávnenú distribúciu. Táto oblasť informatiky je však mladá a je v tieni watermarking-u multimédií, venuje sa jej iba niekoľko vedcov. Cieľom tejto práce bolo vytvoriť prehľad oblasti watermarking-u spustiteľného kódu a vychádzajúc z niektorej existujúcej techniky navrhnúť novú, ktorá by bola odolnejšia voči novooobjaveným útokom.

Obidva ciele práce sme splnili. Preštudovali sme dostupné články a vytvorili sme prehľad kategórií a kritérií kvality techník watermarking-u softvéru, prehľad útokov voči vodoznakom a opísali sme existujúce techniky. Táto časť práce umožní získať prehľad o tejto oblasti dosiaľ nezainteresovanému záujemcovi a podrobnejší opis najnovších techník mu pomôže pochopiť aktuálny stav a prípadne zvoliť smer skúmania.

Vychádzajúc z Dynamic Path-Based Watermarking techniky sme navrhli novú techniku, ktorá vkladá vodoznak do grafu volania funkcií. Vďaka tomu je odolná voči mnohým útokom a v porovnaní s DPB technikou je odolnejšia voči štatistickým a diferenčným útokom. Jej nevýhodou je nižšia kapacita a náročnosť na implementáciu.

Predkladaná práca teda splnila definované ciele a posunula skúmanú oblasť o krok vpred. Možnosť ďalšieho skúmania vidíme v zvyšovaní odolnosti techník voči hybridnému staticko-dynamickému útoku, a to najmä väčším previazaním štruktúr kódujúcich vodoznak s hostiteľským programom.

Literatúra

[1] Christian Collberg, Clark Thomborson: Software Watermarking: Models and Dynamic Embeddings, *POPL'99*, San Antonio, Január 1999

[2] Christian Collberg, Clark Thomborson: Watermarking, Tamper-Proofing, and Obfuscation – Tools for Software Protection, *IEEE Transations on Software Engineering, 28*, August 2002

[3] Patrick Cousot, Radhia Cousot: An abstract interpretation-based framework for software watermarking, *POPL'04*, Benátky, Január 2004

[4] C. Collberg, E. Carter, S. Debray, A. Huntwork, J. Kececioglu, C. Linn, M. Stepp: Dynamic Path-Based Software Watermarking, *SIGPLAN '04, Conference on Programming Language Design and Implementation*, Jún 2004

[5] Amarnath Mullick, Y.N. Srikant: SWuS: Software Watermark using Slices, Technical Report IISc-CSA-TR-2004-7, September 2004

[6] Christian Collberg, Andrew Huntwork, Edward Carter and Gregg Townsend: Graph Theoretic Software Watermarks: Implementation, Analysis, and Attacks, *International Workshop on Information Hiding*, Máj 2004

[7] Geneviève Arboit: A Method for Watermarking Java Programs via Opaque Predicates, *The Fifth International Conference on Electronic Commerce Research*, 2002

[8] Julien P. Stern, Gael Hachez, Francois Koeune, Jean-Jacques Quisquater: Robust Object Watermarking: Application to Code, *International Workshop on Information Hiding*, 1999

[9] Matias Madou, Bertrand Anckaert, Bjorn De Sutter, Koen De Bosschere: Hybrid Static-Dynamic Attacks against Software Protection Mechanisms, *5th ACM Workshop on Digital Rights Management*, November 2005

[10] MOSKOWITZ: Metho for stega-cipher protection of computer code, U.S. Patent 5 745 569, Január 1996

[11] ROBERT L. DAVIDSON, NATHAN MYHRVOLD: Method and system for generating and auditing a signature for a computer program. U.S. Patent 5 559 884, September 1996

[12] MICHIEL RONSSE, JONAS MAEBE, KOEN DE BOSSCHERE: Software instrumentation using dynamic techniques, 2002

[13] ERIC W. WEISSTEIN: Isomorphic Graphs, *MathWorld – A Wolfram Web Resource.* http://mathworld.wolfram.com/IsomorphicGraphs.html

[14] STEVE CORNETT: Code Coverage Analysis, 2004, http://www.bullseye.com/coverage.html

[15] MICHAEL L. FREDMAN, JÁNOS KOMLÓS, ENDRÉ SZEMERÉDI: Storing a Sparse Table with $O(1)$ Worst Case Access Time, *Journal of the ACM, 31 (3)*, 1984

[16] KEN STEVENS A KOL.: An Asynchronous Instruction Length Decoder, *IEEE Journal of Solid-State Circuits, 36(2)*, Február 2001

[17] DENNIS C. LEE, PATRICK J. CROWLEY, JEAN-LOUP BAER, THOMAS E. ANDERSON, BRIAN N. BERSHAD: Execution Characteristics of Desktop Applications on Windows NT, *ISCA'98*, Jún 1998

[18] GINGER MYLES, CHRISTIAN COLLBERG: Software watermarking via opaque predicates: Implementation, analysis, and attacks, *Electronic Commerce Research*, 2006

[19] RAMARATHNAM VENKATESAN, VIJAY VAZIRANI, SAURABH SINHA: A Graph Theoretic Approach to Software Watermarking, *International Workshop on Information Hiding*, Apríl 2001

[20] INGEMAR J. COX, JEAN-PAUL M.G. LINNARTZ: Some general methods for tampering with watermarks, *IEEE Journal on Selected Areas in Communications, 16(4)*, Máj 1998

[21] DAVID AUCSMITH: Tamper Resistant Software: An Implementation, *Information Hiding, First International Workshop*, R.J. Anderson, ed., Máj 1996

[22] NEIL F. JOHNSON: Steganography, http://www.jjtc.com/stegdoc/steg1995.html