Desofuscació de codi mitjançant simplificació assistida per síntesi de programes d'expressions mixtes booleanes-aritmètiques

Arnau Gàmez i Montolio (me@arnaugamez.com)

30 de setembre de 2020

Resum

Aquest projecte estudia el rerefons teòric de les expressions mixtes booleanes-aritmètiques (MBA), així com la seva aplicació pràctica en el camp de l'ofuscació de codi, una tècnica usada tant per les amenaces de programari maliciós com pels sistemes de protecció de programari comercial, per tal de complicar el procés d'enginyeria inversa sobre la totalitat (o parts) d'un programari. Una expressió MBA està formada per operadors aritmètics sobre enters, per exemple $(+, -, *)$ i operadors bit a bit, per exemple $(\wedge, \vee, \oplus, \neg)$. Les expressions MBA es poden aprofitar per ofuscar el flux de dades del codi aplicant iterativament regles de reescriptura i identitats de funcions que compliquen (ofusquen) l'expressió inicial, al mateix temps que se'n preserva el comportament semàntic. Aquesta possibilitat està motivada pel fet que la combinació d'operadors d'aquests dos camps diferents *no interactuen gaire bé*: no tenim regles (distributivitat, factorització...) o una teoria general per tractar amb aquests operadors barrejats. Les tècniques actuals de desofuscació de codi per tractar la simplificació d'aquest tipus d'ofuscació del flux de dades estan limitades pel fet d'estar fortament lligades a la complexitat sintàctica. Explorem nous enfocaments basats en síntesi de programes per tal d'abordar la qüestió de simplificar expressions MBA a partir d'un raonament al voltant del comportament semàntic de les expressions ofuscades, en lloc de centrar-nos en la seva representació sintàctica, tot discutint la seva aplicabilitat i limitacions. Presentem la nostra pròpia eina *r2syntia* que integra *Syntia*, una eina de codi obert per síntesi de programes, en el marc de treball d'enginyeria inversa *radare2* per tal d'obtenir la semàntica de codi ofuscat a partir del seu comportament d'entrada/sortida.

1 Introducció

L'ofuscació de codi és una protecció tècnica contra atacs informàtics emmarcats en l'escenari en què l'atacant disposa d'una instància del programa i controla totalment l'entorn on s'executa. Més formalment, definim l'ofuscació de codi com el procés de transformar un programa P en un altre programa P', funcionalment equivalent però més difícil d'analitzar i d'extreure'n informació, amb la motivació principal de complicar l'enginyeria inversa sobre el mateix programa. Entre els escenaris més notables on es fa servir habitualment destaquen els sistemes de protecció de programari comercial, per tal de salvaguardar la propietat intel·lectual o la gestió de continguts digitals, així com les amenaces de programari maliciós, per tal de guanyar temps de manera inadvertida i evitar-ne la detecció automàtica.

En essència, l'ofuscació de codi consisteix a aplicar una transformació per *embolicar* el flux de control i dades del programa, en diferents nivells d'abstracció (codi font, executable compilat o representació intermèdia) i afectació (programa sencer, funció o bloc base).

Pel que fa a l'ofuscació del control de flux, en destaquem les tècniques següents:

- Predicats opacs: es ramifica codi seqüencial a partir d'una condició (predicat) P creada específicament perquè sempre sigui certa (o falsa).
- Aplanament del flux de control: l'estructura del flux de control d'una funció es reemplaça per un dispensador central.

Quant a les tècniques d'ofuscació del flux de dades, en remarquem les següents:

- Inserció de codi mort: s'afegeix deliberadament un conjunt d'instruccions que no tenen cap efecte sobre el resultat final.
- Substitucions de patrons: se substitueixen instruccions adjacents per d'altres més *complicades* que preserven l'equivalència semàntica.

Pel que fa a la desofuscació de codi, idealment voldríem definir-la com el procés de transformar un programa ofuscat P' en un programa P'' més senzill d'analitzar i el més semblant possible al programa P inicial. Això, però, és molt complicat de garantir per diversos motius: l'analista no té accés al programa P inicial i, a més a més, habitualment està més interessat a entendre parts específiques del programa que no pas a reconstruir tot un programa funcional desofuscat.

2 Expressions mixtes booleanes-aritmètiques (MBA)

Definim formalment una expressió MBA polinòmica E de la manera següent:

$$E = \sum_{i \in I} a_i \left(\prod_{j \in J_i} e_{i,j}(x_1, \ldots, x_t) \right)$$

on la suma i el producte són mòdul 2^n, a_i són constants en $\mathbb{Z}/2^n\mathbb{Z}$, $e_{i,j}$ són expressions bit a bit en les variables $x_1, \ldots, x_t \in \{0,1\}^n$, $I \subset \mathbb{Z}$ i per a tot $i \in I$, $J_i \subset \mathbb{Z}$ són conjunts d'índexs finits.

Observem un exemple d'una expressió MBA polinòmica:

$$E = 8458(x \vee y \wedge z)^3((xy) \wedge x \vee t) + x + 9(x \vee y)yz^3$$

Una simplificació habitual que podem considerar són les expressions MBA lineals, donades per la següent restricció sobre la definició de les expressions MBA polinòmiques:

$$E = \sum_{i \in I} a_i e_i(x_1, \ldots, x_t)$$

A continuació veiem un exemple d'expressió MBA lineal:

$$E = (x \oplus y) + 2 \times (x \wedge y)$$

En aquest cas, observem que l'expressió *simplifica* (és semànticament equivalent) a l'expressió més senzilla $x + y$.

A partir d'aquestes expressions mixtes, disposem de dues tècniques principals per tal d'ofuscar el flux de dades (de parts) del codi d'un programa. Per una banda, aplicant regles de reescriptura que transformaran un operador en una expressió MBA equivalent. D'altra banda, a partir de la inserció d'identitats, on apliquem una funció f i la seva inversa f^{-1} sobre una expressió inicial. Ho il·lustrem en l'exemple següent:

Sigui $E_1 = x + y$ en $\mathbb{Z}/2^8\mathbb{Z}$. Considerem les funcions següents f i f^{-1} en $\mathbb{Z}/2^8\mathbb{Z}$:

$$f : x \mapsto 39x + 23$$
$$f^{-1} : x \mapsto 151x + 111$$

Considerem ara l'expressió e_1 obtinguda a partir d'aplicar a E_1 la regla de reescriptura

$$x + y \mapsto (x \oplus y) + 2 \times (x \wedge y)$$

derivada de l'exemple anterior:

$$e_1 = (x \oplus y) + 2 \times (x \wedge y)$$

A continuació apliquem la inserció d'identitats generada per f i f^{-1}:

$$e_2 = f(e_1) = 39 \times e_1 + 23$$
$$E_2 = f^{-1}(e_2) = 151 \times e_2 + 111$$

Finalment, expandim E_2 per observar l'expressió ofuscada obtinguda a partir de $E_1 = x + y$:

$$E_2 = 151 \times (39 \times ((x \oplus y) + 2 \times (x \wedge y)) + 23) + 111$$

Podem aplicar aquestes tècniques iterativament per tal d'aconseguir una expressió ofuscada arbitràriament complexa, des d'un punt de vista *sintàctic*.

Quant a les tècniques de simplificació teòriques d'expressions MBA, trobem dos vessants i enfocaments clarament diferenciats:

- Trobar una *representació canònica* de les expressions MBA. La idea bàsica és representar les expressions MBA com a expressions booleanes i computar l'efecte de cada operació en cada bit del valor resultant, fent servir la forma normal algebraica per garantir-ne la unicitat. El principal avantatge és que transforma el problema de simplificació d'expressions MBA a un problema de simplificació purament booleana. Tanmateix, el procés de *canonicalització* pot ser molt costós i identificar expressions simbòliques a partir d'expressions canòniques booleanes no és gens trivial. A més, presenta problemes d'escalabilitat amb el tractament d'expressions que involucren una mida gran de bits.
- Trobar una *forma simbòlica equivalent* però més simple. La idea principal és fer servir tècniques de simplificació existents per a les parts de l'expressió que contenen un únic tipus d'operador (aritmètic o bit a bit), complementat per un sistema de reescriptura de termes a partir de regles de reescriptura que podem obtenir invertint la direcció de les regles usades per ofuscació. El principal avantatge és que, com que treballem simbòlicament, aquesta simplificació no està limitada per la dimensió en nombre de bits a tractar i les expressions poden ser representades de manera molt més eficient. Tot i això, veiem que aquest enfocament és molt sensible a la mida de l'expressió ofuscada i extremadament dependent de les regles de reescriptura disponibles.

3 Síntesi de programes

Les tècniques actuals de desofuscació de codi que aborden la simplificació del flux de dades (en particular, simplificació d'expressions MBA) estan fortament limitades, en tant que la seva eficàcia es troba estretament lligada a la complexitat sintàctica del codi analitzat.

Per poder abordar aquest problema de manera més eficaç, ens agradaria poder raonar sobre el comportament semàntic del codi, en lloc de romandre limitats per la seva expressió sintàctica. En aquest sentit, introduïm tècniques de síntesi de programes amb l'objectiu de *sintetitzar* la semàntica d'un fragment particular de codi, presumptament ofuscat. En raonar sobre la semàntica del codi, ja no estem limitats per la complexitat sintàctica del codi subjacent, la qual pot augmentar arbitràriament, sinó només per la seva complexitat semàntica. En presentem un exemple motivador:

Considerem la següent funció que descriu una expressió MBA ofuscada:

$$f(x, y, z) = (((x \oplus y) + ((x \wedge y) \times 2)) \vee z) + (((x \oplus y) + ((x \wedge y) \times 2)) \wedge z)$$

Aleshores, podem tractar f com una *capsa negra* i observar el seu comportament:

$$\begin{aligned} (1,1,1) &\longrightarrow \boxed{f(x,y,z)} \longrightarrow 3 \\ (2,3,1) &\longrightarrow \boxed{f(x,y,z)} \longrightarrow 6 \\ (0,-7,2) &\longrightarrow \boxed{f(x,y,z)} \longrightarrow -5 \\ &\dots \end{aligned}$$

Així, el nostre objectiu és *aprendre* (o *sintetitzar*) una funció més simple amb el mateix comportament d'entrada/sortida (E/S):

$$h(x, y, z) = x + y + z$$

Podem definir la síntesi de programes com el procés de construir automàticament programes que satisfan una especificació donada. Quan parlem d'especificació, ens referim a algun mecanisme per tal d'indicar a l'ordinador *què és el que ha de fer*, tot deixant els detalls d'implementació per al *sintetitzador*. Entre els tipus d'especificacions més habituals trobem:

- Una especificació formal en alguna lògica (p. ex. lògica de primer ordre).
- Un conjunt de parells d'E/S que descriuen el comportament del programa.
- Una implementació de referència (*oracle*) que ens permetrà generar parells d'E/S.

És clar que la naturalesa del nostre problema condueix a un estil de síntesi de programes inductiu i guiat per un oracle. A grans trets, la idea bàsica és fer servir el codi ofuscat com a oracle per generar parells d'E/S i, a continuació, determinar quin és el millor programa candidat, el qual ha de coincidir amb aquest comportament d'E/S.

Pel que fa al treball existent en matèria de desofuscació de codi amb tècniques de síntesi de programes, cal destacar la publicació de *Syntia* (2017), que fa servir un enfocament estocàstic, convertint el problema de trobar un programa candidat en un problema d'optimització estocàstica: a cada iteració es generen resultats intermedis que evolucionen cap a l'òptim global (el *millor candidat*) guiats per una funció de cost.

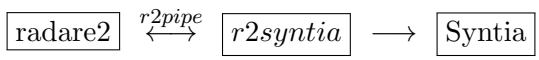
Quant a les limitacions d'aquests tipus d'enfocaments guiats per un oracle d'E/S, trobem:

- Codi amb una alta complexitat semàntica.
- Codi no-determinista.
- Funcions de punt (retornen el mateix valor per totes les entrades excepte per una diferencial).

4 Implementació de r2syntia

Pel vessant pràctic del treball, he desenvolupat una eina anomenada *r2syntia*, la qual integra la funcionalitat de síntesi de programes de *Syntia* amb l'entorn d'anàlisi i emulació de codi proporcionat pel marc de treball d'enginyeria inversa *radare2*.

El funcionament essencial de *r2syntia* és el següent: podem invocar *r2syntia* des d'una sessió activa de *radare2* on estiguem realitzant l'anàlisi de l'executable que conté codi ofuscat. Llavors, *r2syntia* se serveix de les funcionalitats d'emulació de codi de *radare2* (fent servir *r2pipe*, el mecanisme de comunicació intern que proporciona *radare2* per interactuar amb ell des d'altres llenguatges de programació) per tal de generar parells d'E/S per a les variables (registres del processador i adreces de memòria) especificades en la crida. A partir dels parells d'E/S generats, *r2syntia* invoca *Syntia* per tal de sintetitzar el comportament semàntic de la variable de sortida respecte a les variables d'entrada.



5 Conclusions

L'objectiu principal d'aquest treball era analitzar les tècniques actuals de desofuscació de codi, posant especial èmfasi en els procediments de síntesi de programes que aborden la desofuscació del flux de dades a partir de la simplificació d'expressions MBA. Com a objectiu secundari, volíem contribuir amb propostes pràctiques, provar-les i validar-les en un entorn controlat. Hem assolit amb èxit aquests objectius mitjançant un conjunt d'assoliments específics:

- Hem analitzat i presentat tècniques comunes d'ofuscació de codi utilitzades en amenaces de programari maliciós i sistemes de protecció de programari comercial, aportant idees bàsiques per avaluar-ne la qualitat, així com alguns aspectes teòrics i pràctics a tenir en compte a l'hora d'abordar la desofuscació de codi.
- Hem realitzat un desenvolupament teòric de les expressions MBA, presentant les principals tècniques d'ofuscació de flux de dades a partir de transformacions d'expressions MBA semànticament equivalents.
- Hem estudiat la viabilitat dels enfocaments de síntesi de programes per ajudar en el procés de desofuscació del codi.
- Finalment, la nostra implementació de *r2syntia* és, fins on sabem, la primera integració d'una eina (i enfocament) de síntesi de programes amb un marc de treball d'enginyeria inversa.