Maciej BRZOZOWSKI, Vyacheslav N. YARMOLIK
POLITECHNIKA BIAŁOSTOCKA, WYDZIAŁ INFORMATYKI

Obfuskacja - narzędzie zabezpieczające prawa autorskie do projektów sprzętowych

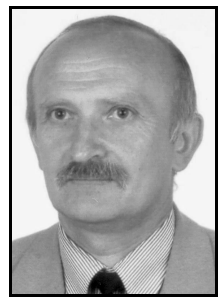
Mgr inż. Maciej BRZOZOWSKI

Ukończył studia na Wydziale Informatyki Politechniki Białostockiej w 2005. Od 2005 jest asystentem na wydziale Informatyki Politechniki Białostockiej. Specjalizuje się w zabezpieczaniu praw autorskich do projektów sprzętowych. Prowadzi badania nad wykorzystaniem znaków wodnych (watermarking) oraz obfuskacji w ochronie własności intelektualnej.

e-mail: brzozowski@ii.pb.bialystok.pl

Prof. dr hab. inż. Vyacheslav N. YARMOLIK

Ukończył studia na Białoruskim Państwowym Uniwersytecie Informatyki i Radioelektroniki w Mińsku w 1973. W 1979 obronił pracę doktorską, a w 1990 uzyskał tytuł doktora habilitowanego. Od 34 lat pracuje w dziedzinie kryptografii oraz testowania pamięci.



e-mail: yarmolik@ii.pb.bialystok.pl

Streszczenie

Oprogramowanie jest coraz częściej rozpowszechniane w postaci kodu źródłowego. W niezabezpieczonym kodzie łatwo jest dokonać modyfikacji a następnie włączyć w „nowej" formie w inny projekt (produkt). Powstało wiele technik i narzędzi zabezpieczających przed kradzieżą oprogramowania takich jak znaki wodne lub odciski palca [1, 2]. Jedną z metod ochrony praw autorskich, zabezpieczającą przeciwko nieautoryzowanemu wykorzystaniu kodu, jest obfuskacja. Proces ten sprawia, że kod źródłowy staje się nieczytelny i trudny w analizie aczkolwiek nadal posiada wszystkie poprzednie właściwości funkcjonalne. W artykule przedstawimy techniki obfuskacji dla sprzętowego języka VHDL (Very High Speed Integrated Circuit Hardware Description Language) oraz przykłady ich zastosowania w ochronie praw autorskich ze zwróceniem szczególnej uwagi na optymalność zabezpieczonego kodu źródłowego.

Słowa kluczowe: obfuskacja, ochrona własności intelektualnej, VHDL, optymalizacja projektu.

Obfuscation – tool for protecting intelectual property for design

Abstract

Software is more and more frequently distributed in form of source code. Unprotected code is easy to alter and build in others projects. One of the method for protection against attacks on intellectual rights is obfuscation, a process that makes software unintelligible but still functional. In this paper we review several generic techniques of obfuscation VHDL (Very High Speed Integrated Circuit Hardware Description Language) code and present a set of them for designs protection witch focusing on code optimalization. We are going to describe some related experimental work.

Keywords: obfuscation, intelectual property protection, VHDL, design optymalization.

1. Wstęp

Zaciemnianie kodu (obfuskacja, z ang. obfuscation) jest techniką przekształcania kodu źródłowego oprogramowania, która zachowuje jego działanie semantyczne, ale znacząco utrudnia analizę i zrozumienie. Obfuskacja jest więc zamierzonym działaniem, mającym na celu ochronę własności intelektualnej przy zachowaniu pełnej funkcjonalności oprogramowania. Ujmując rzecz bardziej ogólnie obfuskacja jest próbą ukrycia sposobu działania zabezpieczanego programu.

Jako pierwszy definicję obfuskacji podał Christian Collberg w „A taxonomy of obfuscating transformations" [3] używaną i rozwijaną w późniejszych publikacjach. Gaël Hachez w zbiorczej pracy „A Comparative Study of Software Protection Tools…" [4] zauważa, że definicja przyjęta przez C. Collberga daje możliwość wprowadzenia w błąd użytkownika oprogramowania dlatego zmodyfikował ją na:

$P \xrightarrow{\tau} P'$ jest transformatą obfuskacyjną jeżeli:

a) program P przerywa swoją pracę lub kończy ją z błędami program P' musi też przerywać swoje działanie lub skończyć je z błędami.
b) w przeciwnym przypadku P' musi zakończyć swoje działanie i wygenerować identyczne wyniki co P.

W tym miejscu należy zauważyć ograniczenia jakim podlega obfuskacja [5]. Należy pamiętać, że każdy program P' będzie można odtworzyć do P'', który swoją budową będzie bardzo zbliżony do programu P. Działanie tego typu pochłonie dużą ilość czasu oraz ogromny nakład środków, lecz zawsze będzie możliwe do wykonania. Dlatego zadaniem obfuskacji nie jest zabezpieczenie programu przed jego dekompilacją, lecz w znaczącym stopniu wydłużenie czasu potrzebnego na analizę i zrozumienie jego działania.

```
int i;main(){for(;i["]<i;++i){--i;}"];read('-'-'-',i+++"hell\o,
world!\n",'/'/'/'));}read(j,i,p){write(j/p+p,i---j,i/i);}
```

Listing. 1. Zwycięzca 1st International Obfuscated C Code Contest - w kategorii Dishonorable mention – 1984
Listing. 1. Winner of 1st International Obfuscated C Code Contest – in category Dishonorable mention – 1984

Na listingu 1 przedstawiono kod źródłowy programu anonimous.c zwycięzcy 1st International Obfuscated C Code Contest 1984 w kategori Dishonorable mention (autor nieznany). „Zaciemniony" przykład pokazuje w jakim stopniu można utrudnić analizę krótkiego kodu. Dla osób, które chciałyby podjąć się analizy kodu i sprawdzić jej poprawność zamieszczamy pierwotny kod (listing 2).

```
#include<stdio.h>int main(void){    printf("Hello World!\n");    return 0;}
```

Listing. 2. Hello World!
Listing. 2. Hello World!

Obecnie znana jest szeroka gama technik obfuskacyjnych jednak ich zastosowanie nie zostało przeniesione na pole języków programowania sprzętowego. Możliwości technik zaciemniania kodu przedstawimy na przykładzie języka VHDL (Very High Speed Integrated Circuit Hardware Description Language).

2. Optymalizacja obfuskacji

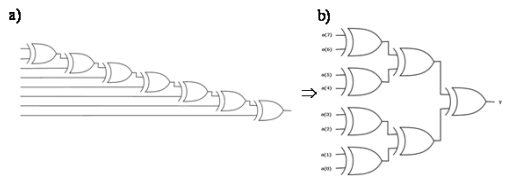
W projektach sprzętowych, w przeciwieństwie do projektów programistycznych [6, 7], niezwykle istotną role odgrywa optymalność projektu po zastosowaniu technik obfuskacyjnych. Nieodpowiedni dobór transformat może w znacznym stopniu obniżyć wydajność projektu, a w skrajnych przypadkach, całkowicie uniemożliwić jego dalsze wykorzystanie. Takie

zachowanie projektu może wynikać ze zbyt dużego wzrostu obszaru zajmowanego na układzie docelowym przez projekt lub nadmierny wzrost czasu propagacji na krytycznej ścieżce (*critical path*). W poprzednich pracach [6, 7] przedstawiliśmy szereg technik obfuskacyjnych zasymilowanych dla zastosowania w języku VHDL – analizę przeprowadziliśmy pod względem wzrostu obszaru zajmowanego przez projekt oraz maksymalnego taktowania układu po zastosowaniu transformat.

a)
```
process (x)
   variable tmp : std_logic;
begin
   tmp = x(7);
   for i in 6 downto 0 loop
   tmp:=tmp xor x(i);
   end loop;
   y<=tmp;
end process;
```
⇒ b)
```
l<= (x(7) xor x(6)) xor (x(5) xor x(4));
r<= (x(3) xor x(2)) xor (x(1) xor x(0))
y<=l xor r;
```

Listing. 3. Kontroler parzystości w języku VHDL a) kaskada b) drzewo
Listing. 3. Parity controler in VHDL language a) cascade b) tree

Do przeanalizowania wpływu implementacji układu na prędkość jego działania posłużymy się prostym przykładem akademickim. Na listingu 3 przedstawiono dwie możliwe implementacje układu kontrolera parzystości dla języka VHDL. Oba podejścia przedstawiają układy których wyniki działania będą identyczne. Jedyną różnicą będzie ich fizyczne rozmieszczenie na układzie (rysunek 1). Symbol ⇒ oznacza modyfikację kodu źródłowego.



Rys. 1. Kontroler parzystości schemat RTL; struktura a) kaskadowa b) drzewa
Fig. 1. RTL scheme of paritty controler a) cascade b) tree structure

O ile niewielka różnica w rozmieszczeniu bramek nie pociąga za sobą zmiany zajmowanego obszaru przez projekt to w dużym stopniu zwiększa prędkość działania układu poprzez zmniejszenie czasu propagacji na krytycznej ścieżce. Zakładając czas propagacji bramki *xor* równy T_{xor} czas propagacji dla układu o strukturze kaskadowej będzie wynosił $T_K = T_{xor} \cdot (n-1)$ a dla układu o strukturze drzewiastej odpowiednio $T_D = T_{xor} \cdot \lceil \log_2 n \rceil$. Przy ośmiu wejściach $n = 8$ wydajność układu kaskadowego w stosunku do drzewiastego

$$\frac{T_K}{T_D} = \frac{T_{xor} \cdot (n-1)}{T_{xor} \cdot \lceil \log_2 n \rceil} = \frac{n-1}{\lceil \log_2 n \rceil} = \frac{7}{3} \approx 2{,}3, \qquad (1)$$

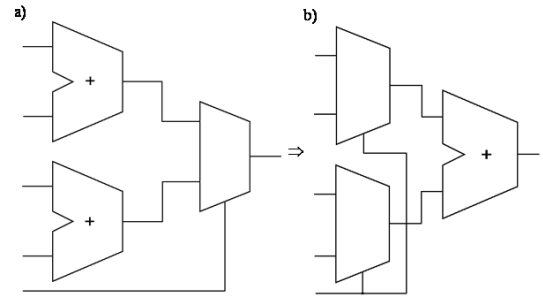
zwiększa się ponad dwukrotnie. W pierwszym przypadku na krytycznej ścieżce propagacji znajduje się aż siedem bramek *xor* natomiast dla przypadku drugiego jest ich tylko trzy. Powyższy przykład pokazuje jak duże znaczenie ma odpowiedni zapis kodu przed jego syntezą by uzyskać jak najlepsze czasy propagacji.

W kolejnym przykładzie przeanalizujemy układ sumatora. Na listingu 4a przedstawiono prosty kod przypisania sumy wektorów uwarunkowanych wyrażeniem logicznym s.

Po pobieżnej analizie może wydawać się, że do takiego zapisu nie można mieć zastrzeżeń - funkcjonalnie działa on poprawnie - jednak jeśli przeanalizuje się schemat RTL komponentu można zauważyć nadmiarowość przy wykorzystaniu zasobów układu.

a)
```
y<=a+b when s='1' else c+d;
```
⇒ b)
```
tmp1<= a when s='1' else c;
tmp2<= b when s='1' else d;
y<=tmp1+tmp2;
```

Listing. 4. Sumator I
Listing. 4. Adder unit (ver. I) in VHDL language



Rys. 2. Sumator schemat RTL a) przed b) po optymalizacji
Fig. 2. RTL scheme of adder unit a) before b) after optymalization

Listing 4b przedstawia układ po optymalizacji. Kod komponentu stał się mniej zrozumiały i wymaga poświęcenia większej ilości czasu na analizę, niż kod z listingu 4a. Jeżeli przeanalizujemy rysunek 2b, łatwo zauważymy, że zmniejszyła się ilość wykorzystywanej powierzchni na układzie docelowym. Podobna sytuacja zachodzi na kolejnym przykładzie (listingu 5).

a)
```
y<=a+b when s='1' else a+c;
```
⇒ b)
```
tmp<= b when s='1' else c;
y<=a+tmp;
```

Listing. 5. Sumator II
Listing. 5. Adder unit (ver. II) in VHDL language

Po zoptymalizowaniu liczba sumatorów w układzie zmniejsza się z dwóch do jednego.

3. Zwiększanie wydajności projektu a obfuskacja

Powyżej rozważane przykłady odnosiły się w głównej mierze do przetwarzania równoległego. Na listingu 6 zamieszczono kod przetwarzania sekwencyjnego na przykładzie cyklicznego odczytu wartości z pamięci ROM.

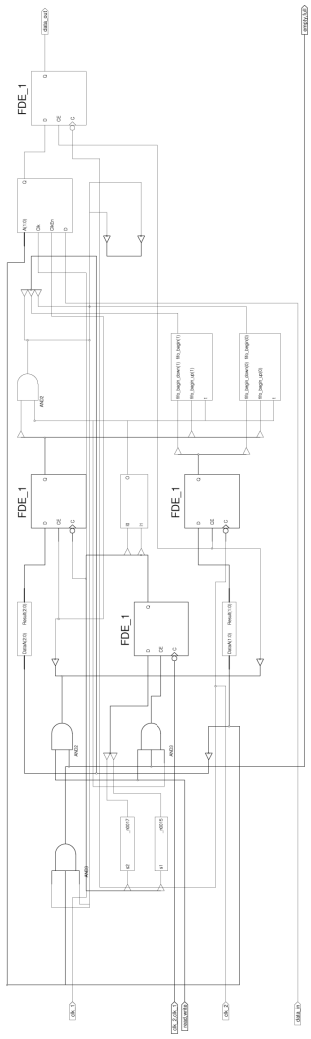
a)
```
signal i:natural range 0 to n-1;

process (clk)
begin
  if clk'event and clk='1' then
     y<=rom(i);
     i<=i+1;
  end if;
end process;
```
⇒ b)
```
y_next<=rom(i);
i_next<=i+1;
process (clk)
begin
   if (...) then
      y<=y_next;
      i<=i_next;
   end if;
end process;
```
c)
```
rom_next<=rom(0)&rom(n-1 downto 1);
y_next<=rom(0);

process (clk)
⇒begin
    if clk'event and clk='1' then
       y<=y_next;
       rom<=rom_next;
    end if;
end process;
```

Listing. 6. Cykliczny odczyt z pamięci ROM
Listing. 6. Cyclic ROM reader in VHDL language

Kod źródłowy z listingu 6a został zmodyfikowany w dwóch krokach. W pierwszym pod względem wydajności listing 6b. W kolejnym kroku (listing 6c) zoptymalizowano miejsce wykorzystywane przez projekt na układzie docelowym poprzez usunięcie nadmiarowej liczby multiplekserów. W przypadku zabezpieczania kodu o wysokim stopniu optymalności, gdzie ciężko jest zmodyfikować projekt tak by nie obniżać parametrów jego działania, zawsze możliwe jest użycie technik obfuskacyjnych z grupy zaciemniania układu strony (ang. layout obfuscation) [3].

4. Wnioski

Największym ograniczeniem obfuskacji dla języków sprzętowych jest czas reakcji układu na zmieniające się sygnały wejściowe – maksymalny czas propagacji sygnału dla układów kombinacyjnych lub maksymalna częstotliwość taktowania dla układów sekwencyjnych – oraz obszar zajmowany na układzie zabezpieczeniu projektu. Większość technik zaciemniania kodu nie daje się zasymilować z innych, nie sprzętowych, języków programowania ze względu na nadmierny wzrost wyżej wymienionych parametrów.



Rys. 3. Schemat RTL kolejki FIFO
Fig. 3. RTL scheme of dual clock FIFO

Na rysunku 3 przedstawiliśmy schemat RTL zabezpieczonego technikami obfuskacyjnymi projektu kolejki FIFO taktowanej dwoma zegarami (odczyt, zapis). Projekt po zabezpieczeniu nie zmienił swoich właściwości funkcjonalnych.

Na podstawie przeanalizowanych transformat oraz danych zawartych w poprzednich artykułach [6,7] powstaje oprogramowanie ExpressObfuscation, którego zadaniem jest zabezpieczania kodu źródłowego języka VHDL.

Należy pamiętać, że zabezpieczanie kodu projektu nie jest jednoznaczne ze spadkiem jego wydajności. Odpowiedni dobór zbioru transformat dopasowanych do właściwości projektu nie spowodują obniżenia jego parametrów pracy a w niektórych przypadkach mogą zwiększyć jego wydajność.

Artykuł powstał w ramach pracy statutowej S/WI/6/2008 Politechniki Białostockiej.

5. Literatura

[1] Johnson N. F., Duric Z., Jajodia S.: Information hiding – steganography and watermarking – attacks and countermeasures, Kluwer Academic Publishers, Norwell 2001
[2] Katzenbeisser S., Petitcolas F. A. P.,: Information hiding – techniques for steganography and digital watermarking, Artech House, Norwood 2000
[3] C. Collberg, C. Thomborson, and D. Low. “A Taxonomy of Obfuscating Transformations”, Technical Report 148, Department of Computer Science. The University of Auckland, July 1997
[4] Gaël Hachez. “A Comparative Study of Software Protection Tools Suited for E-Commerce with Contributions to Software Watermarking and Smart Cards”, PhD thesis, Universite Catholique de Louvain, March 2003
[5] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan and K. Yang. “On the (Im)possibility of Obfuscating Programs” In J. Kilian, editor, Advances in Cryptology - CRYPTO '01, volume 2139 of Lectures Notes in Computer Science (LNCS), pages 1-18. Springer-Verlag, 2001.
[6] Maciej Brzozowski, Vyacheslav N. Yarmolik "VHDL obfuscation techniques for protecting intellectual property rights on design", 5th IEEE East-West Design and Test Symposium, Yerevan, 2007, p.371-375
[7] Maciej Brzozowski, Vyacheslav N. Yarmolik "Obfuscation as Intellectual Rights Protection in VHDL Language", 6th Computer Information Systems and Industrial Management Applications, Los Alamitos, 2007, p.337-340
[8] D. Low, “Java Control Flow Obfuscation. Master's thesis”, Department of Computer Science, University of Auckland, June 1998
[9] Gregory Wroblewski, “General Method of Program Code Obfuscation”, PhD Dissertation, Wroclaw University of Technology, Institute of Engineering Cybernetics, 2002
[10] Christian S. Collberg, Clark D. Thomborson, and Douglas Low. Breakingabstractions and unstructuring data structures. In International Conferenceon Computer Languages (ICCL), pages 28–38, 1998.
[11] Maciej Brzozowski, Vyacheslav N. Yarmolik "System wbudowujący znak wodny w kod źródłowy oprogramowania" Modelowanie i symulacja komputerowa w technice : IV Sympozjum, Łódź, Wyższa Szkoła Informatyki (Łódź, Polska),Łódź, 2005
[12] Maciej Brzozowski, Vyacheslav N. Yarmolik "Analiza statystycznych właściwości kodu źródłowego", XI Warsztaty Naukowe PTSK : Symulacja w badaniach i rozwoju : zbiór referatów, PTSK 2004
[13] Maciej Brzozowski, Vyacheslav N. Yarmolik "Metody i implementacja ochrony praw autorskich w oprogramowaniu komercyjnym", Wiadomości Elektrotechniczne, nr 12 (2004), s.28-29
[14] Thomas J. McCabe. A complexity measure. IEEE Transaction on Software Engineering, 2(4):308–320, 1976.
[15] C. Collberg and C. Thomborson, “Watermarking, TamperProofing, and Obfuscation --- Tools for Software Protection”, IEEE Transactions on Software Engineering, Vol. 28, No. 8, August 2002.
[16] C. Collberg and C. Thomborson, “On the limits of Software Watermarking”, Technical Report #164, Department of Computer Science, The University of Auckland, August 1998
[17] Jasvir Nagra and Clark Thomborson and Christian Collberg, “A Functional Taxonomy for Software Watermarking”, Twenty-Fifth Australasian Computer Science Conference (ACSC2002),ACS, Melbourne, Australia, 2002
[18] Tapas Sahoo and Christian Collberg. “Software watermarking in the frequency domain: Implementation, analysis, and attacks”, Technical Report TR04-07, Department of Computer Science, University of Arizona, March 2004
[19] Chenxi Wang, A Security Architecture for Survivability Mechanisms, PhD Dissertation, University of Virginia, Department of Computer Science, October 2000
[20] Christian Collberg, Clark Thomborson, Douglas Low, “Manufacturing Cheap, Resilient, and Stealthy Opaque Constructs”, SIGPLAN-SIGACT POPL'98, ACM Press, San Diego, CA, January 1998

Artykuł recenzowany