Analyzing Functional Entropy of Software Intent Protection Schemes

J. Todd McDonald
Eric Trias
Alan Lin

Center for Cyberspace Research
Department of Electrical and Computer Engineering
Air Force Institute of Technology
Wright Patterson AFB, OH
Why Do We Try to Protect Software?

Because Programs are Attacked….

- Protect Integrity
  - Decomposing/reusing code
  - Adding new functionalities
- Protect Intent
  - Alter existing functionality
  - Prevent “gaming” functionality
  - Prevent countermeasures
- Protect Ownership/Intellectual Property
- Protect Troops/Mission

We are concerned primarily with software-only means of protection.
Underlying Goals

- Given the hardware/physical environment: make it hard for an adversary to reliably or predictably recover an intermediate or original form (Netlist, source level program code)

- Given recovery of some or all of the intermediate / original description of a circuit or program: make it hard for an adversary to recover, predict, subvert, or copy functionality
Ideal Software Protection…

Is a “Virtual” Black Box possible??

Program Source Code (Java/C++/C)
Circuit Netlist (VHDL/Verilog/BENCH)

Assembly
Realized Circuit/FPGA

INPUT

OUTPUT
Real-world Software Protection…

Program Source Code
(Java/C++/C)

Circuit Netlist
(VHDL/Verilog/BENCH)

Assembly
Realized Circuit/FPGA

INPUT

OUTPUT
General Intuition and Hardness of Obfuscation

Develop America's Airmen Today ... for Tomorrow

The ONLY true “Virtual Black Box”

X1  X2  X3  | 4  | 5  | Y6  | Y7  
0 0  0 | 0  | 0  | 0   | 1
0 0  1 | 0  | 0  | 1   | 1
0 1  0 | 0  | 0  | 0   | 1
0 1  1 | 1  | 1  | 0   | 1
1 0  0 | 0  | 1  | 0   | 1
1 0  1 | 0  | 1  | 1   | 0
1 1  0 | 0  | 1  | 0   | 1
1 1  1 | 1  | 1  | 0   | 1

“The How”

X1  X2  X3  | Y6  | Y7  
0 0  0 | XOR(4,3) | NAND(5,6)
0 0  1 | 1       | 1
0 1  0 | 0       | 1
0 1  1 | 0       | 1
1 0  0 | 0       | 1
1 0  1 | 1       | 0
1 1  0 | 0       | 1
1 1  1 | 0       | 1

Semantic Behavior
Program Understanding

- Adversary’s ability to anticipate a program’s operational manifestation(s)
- Adversary’s ability to gain intent indications by comparing the obfuscated code, or segments, to known code libraries
- Adversary’s knowledge gained relative to the theoretical Virtual Black Box
- Adversary’s ability to extract the information content as manifested in the black box and white box aspects of program code

This is not the same as VBB or hiding all information…
Program Understanding

• **Our Context:** Prevent program understanding by limiting the amount of information gained by an adversary from either the blackbox or whitebox characteristics of a program/circuit
  • Programs are no more than a special information class with well-defined syntax and semantics
  • **Scrambling techniques are limited** because final form of program must adhere to rigid syntax and semantics
  • Program code information content is otherwise equivalent to information content in any other type of bit stream

• **Our Premise:** Program code that is statistically indistinguishable from a random bit stream has negligible information content
Defining Intent Protection

Is there an alternate (or better) way to measure security or protection?

If the adversary cannot determine the function/intent of the device by input/output analysis, we say it is **black-box protected**

If the adversary cannot determine the function/intent of the device by analyzing the structure of the code, we say it is **white-box protected**

**Intent Protected:** Combined black-box and white-box protection does not reveal the function/intent of the program
Random Programs/Circuits

**Goals:**
Can we make **input/output** look random?
Can we make **structure** look random?

Instead of measuring security based on *leakage of information from the obfuscated program*, can we appeal to **entropy** or **randomness** as a measure for confusion in the obfuscated program?
RPM: Random Program Model

Develop America's Airmen Today ... for Tomorrow

\[ P \in \delta_p \]

\[ P' \in \delta_{p'} \]

Indistinguishable(?)

Program family
[Inputs/Outputs/Size/Ω]

\[ \delta_p \subset \delta \]

\[ P_R \in \delta \]
Combined Experimental Framework

**Black-box Refinement**

- Input: $x$
- Program: $p$
- Output: $y$
- Transformation: $s(p, k, X, Y) = q, p''', X', Y''$
- Resolution: $y = q(y', k')$

**Semantic Transformation**

- Input: $x'$
- Program: $p''$
- Transformation: $s(p, k) = r, p'$
- Recovery: $y'' = r(y', k')$

**White-box Randomization**

- Program: $p'$
- Transformation: $s(p''), k) = p'$

**Goals:**
- Can we make input/output look random?
- Can we make structure look random?
Protecting Black-Box Intent

Definition 1. Program $P \rightarrow \{X,Y\}$ is black-box understandable if and only if, given an arbitrarily large set of pairs $IO = (x_i, y_i)$ such that $y_i = P(x_i)$ and $y_j$ an arbitrary element of $Y$ with $(x_j, y_j)$ not an element of $IO$, an adversary can efficiently guess $x_j$ such that $y_j = P(x_j)$ with greater than negligible probability. Otherwise, we say $P$ is black-box obfuscated.
Semantic Transformation

Semantically secure data encryption algorithms are black-box intent protected (BBIP)

Compositions of programs with semantically strong algorithms are likewise BBIP
Protecting Black-Box Intent

Black Box Refinement

\[ x \in X \]
\[ y \in Y \]
\[ x' \in X' \]
\[ y' \in Y' \]

Input \( x \)

Transformation

\[ s(p, k, X, Y) = q, p', X', Y' \]

Program \( p \)

Output \( y \)

Resolution

\[ y' = q(y; k^{-1}) \]

Program \( p' \)

Output \( y' \)
Protecting White-Box Intent

Ideal for AT applications where we shield hardware internals with some (reasonably) trusted AT method

Circuit P’
**Definition 2.** Given access to a random program oracle which transforms any program $p$ using algorithm $E(p)$ into an encrypted version $p'$, and given full access to any encrypted program $p'_{n}$: After knowing any $n$ pairs of original and encrypted programs $\{(p_{1}, p'_{1}), (p_{2}, p'_{2}), ..., (p_{n-1}, p'_{n-1}), (p_{n}, p'_{n})\}$, an adversary that supplies a subsequent program $p_{n+1}$ will receive $p_{n+1}'$ from the oracle which is either: a random program ($P_{R}$) or the encrypted version of the program $p_{n+1}' = E(p_{n+1})$. The program $E(p)$ provides white-box protection if and only if the probability that an adversary is able to distinguish the encrypted program $(p_{n+1}')$ from a random program ($P_{R}$) is $\frac{1}{2} + \varepsilon$, where $\varepsilon$ is a negligible constant.

**Definition 3.** Program $P$ is *intent protected* if and only if it is protected against black-box analysis and white box analysis.
• Conjecture when using Semantic Transformation:
  • If the output bits are predictable, then the output may be predictable
  • Treat each output position as a bit string generator
    • Run statistical randomness tests on each bit

• Questions of Interest to the Random Program Model:
  • Does structural randomness produce functional randomness?
    • Frequency of signature collisions (identical output patterns)
    • Approximate entropy of output bits
  • How random are the output bits?
    • Randomness values for specific statistical test
Methodology

Develop America's Airmen Today ... for Tomorrow

- Unable to evaluate structure with agreed security metrics\(^1\)
- Random Oracle used in absence of a defined security model
  - Sanity check for implementation
- Goal
  - \(O(P)\) structurally and functionally looks like \(P_R\)

\(^1\)National Institute of Standards and Technology
Required components
1. Design control/benchmark programs (deterministic)
2. Generate $P_R$
3. Black-box protect $P$

Analysis
- Compare $P$, black-box protected $P$, and $P_R$
Experimental Design

- Emulate deterministic functions w/combinational circuits
  - Abstracts high-level structure (ISCAS-85)
  - Build random circuits to analyze random circuit properties
  - Parameters (from benchmarks)
    - Input size (in bits)
    - Output size (in bits)
    - Number of intermediate nodes (represents structure)
    - Gate basis: AND, OR, XOR, NAND, NOR, NXOR
Experimental Design

- Goal
  - \(O(P)\) produces random functionality

- Add black-box protection to \(P\)
  - Weaken VBB function preservation
  - Strengthen overall security
    - Accounts for clean-room reverse engineering\(^1\)

- Black-box w/ symmetric key cryptography
  - Produces blocks of pseudo-random bits
  - Pseudo-randomness measures exist\(^2\)

---

1. Schwartz “Reverse Engineering”
2. National Institute of Standards and Technology
Experimental Design

- Configurations
  - Trusted-host output recovery
    - Secure execution on malicious host
    - Full structure; no functionality
  - Output recovery for malicious user
    - Partial execution on malicious host
    - Partial structure; full functionality
    - Loss of generality in function type \( y = a \times b + c \)
  - Full ownership by malicious user/host
    - Secure structural components
    - Full structure; full functionality
    - Software Watermark

- Common design
  - Two-level structural configurations
    - Function Table (FT)
    - Boolean Equation Sets (BES)

<table>
<thead>
<tr>
<th>x</th>
<th>y = f(x + x)</th>
<th>y = f(2x)</th>
<th>y = f(x &lt;&lt; 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( x_m )</td>
<td>( y_m )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>x</th>
<th>y = e(x, k)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>66E94BD…89E0</td>
</tr>
<tr>
<td>1</td>
<td>58E2FCC…455A</td>
</tr>
<tr>
<td>2</td>
<td>F795AAA…C1E0</td>
</tr>
<tr>
<td>3</td>
<td>0388DAC…FE78</td>
</tr>
<tr>
<td>4</td>
<td>8ADE7D8…0291</td>
</tr>
<tr>
<td>5</td>
<td>95B84D1B…89E0</td>
</tr>
<tr>
<td>6</td>
<td>C94DA219…88F2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>( x_n )</td>
<td>( y_n )</td>
</tr>
</tbody>
</table>
Experimental Design

- Complexity based brute-force attack on I/O size
  - Compute all function tables of size $m$-inputs, $n$-outputs
    - Super-exponential process, $O(m^n)$
  - Pair combinations of generated function tables in $m,n$
    - Factorial process, $O(n!)$
  - All operations of a lookup are the same
    - Index search, $O(1)$ [or $O(n)$ for BES]
    - No side-channel (performance/cost) leakage

- Memory size
  - Function tables are at least ($n$-input) * ($m$-output) bits
  - Boolean equation sets are at least ($m$-output) * $p$ terms
    - $m$ equations stored in text form of $p$ terms
    - Exponential size increase $n^{pm}$
Results and Analysis

Develop America's Airmen Today ... for Tomorrow

Approximate Entropy

<table>
<thead>
<tr>
<th>% of flipped output bits</th>
<th>AES</th>
<th>5-128-500 (GC)</th>
<th>5-128-1000 (GC)</th>
<th>c17 no pad</th>
<th>c17 CTR pad</th>
<th>c17 INPUT pad</th>
</tr>
</thead>
<tbody>
<tr>
<td>Approx. Entropy</td>
<td>0.50</td>
<td>0.26</td>
<td>0.26</td>
<td>0.26</td>
<td>0.50</td>
<td>0.50</td>
</tr>
</tbody>
</table>
Results and Analysis

Develop America's Airmen Today ... for Tomorrow

Air University: The Intellectual and Leadership Center of the Air Force
Integrity - Service - Excellence

\[ y = a \times b + c \text{ (CTR)} \]

<table>
<thead>
<tr>
<th></th>
<th>% of 1's</th>
<th>Longest 1's Runs</th>
<th>Excursions from Zero</th>
<th>Zero Cycles</th>
<th>Excusion States</th>
<th>Runs of 1's</th>
</tr>
</thead>
<tbody>
<tr>
<td>y = a \times b + c (CTR)</td>
<td>0.12</td>
<td>51.47</td>
<td>1045.35</td>
<td>0.00</td>
<td>1043.59</td>
<td>639.31</td>
</tr>
<tr>
<td>12-8-100 (CTR)</td>
<td>0.01</td>
<td>22.18</td>
<td>33.75</td>
<td>4.18</td>
<td>32.45</td>
<td>13.27</td>
</tr>
<tr>
<td>12-8-300 (CTR)</td>
<td>0.01</td>
<td>19.05</td>
<td>28.94</td>
<td>1.77</td>
<td>28.56</td>
<td>10.42</td>
</tr>
<tr>
<td>12-8-500 (CTR)</td>
<td>0.01</td>
<td>19.06</td>
<td>22.27</td>
<td>1.66</td>
<td>22.72</td>
<td>7.94</td>
</tr>
<tr>
<td>12-8-100 (CTR) 0001</td>
<td>0.31</td>
<td>1184.28</td>
<td>1441.95</td>
<td>0.00</td>
<td>1430.44</td>
<td>458.58</td>
</tr>
</tbody>
</table>

Std Dev of Tests Across Output Bits
Results and Analysis

- Possible signatures is $(2^{\text{output\_bits}}) \land (2^{\text{input\_bits}})$
  - Collisions occur at ↑ frequency with ↓ intermediate node size
  - Collisions occur at ↑ concentration with ↑ intermediate node size

![Collisions vs. Intermediate Nodes](image1)

![Signature Collisions in 5-2-X](image2)
Conclusions

Develop America's Airmen Today ... for Tomorrow

• Black-box metrics indicate bits where structural entropy is most needed if we keep function preservation property
  • Structural entropy may be insufficient depending on output pattern
  • Smaller circuits are better choices for random selection

• Enumeration is required—larger $n$ requires greater resources upon generation, not execution
  • Advantage to developers with large computational resources
  • Reuse encryption function tables
  • Brute-force attack limited to adversaries with sufficient resources
  • Input/output size is easier to determine than function family
Sponsor

Research sponsorship by:

Air Force Office of Scientific Research (AFOSR)
Information Operations
Questions