DFITS: Data Flow Isolation Technology for Security

Ryotaro Hayashi, Fukutomo Nakanishi, Hiro Yoshi Haruki, Yurie Fujimatsu, Miki Hashimoto
Corporate Research & Development Center, TOSHIBA Corporation

(ryotaro.hayashi, fuk.nakanishi, hiro.yoshi.haruki, yurie.fujimatsu, miki.hashimoto)@toshiba.co.jp

Background & Motivation

• Software protection
  – Protect (the confidential data of) the software from both reading and falsifying by malicious users
  – Guarantee the correct behavior of the software
• Operating system may be cracked or compromise with malware
• Secure processor (XOM[1], AEGIS[2], LMSP[3], etc.)
  – Provides a protected memory area for confidential data, on which the confidential data of the software is encrypted and MAC is generated, and cannot be read and falsified from other entities
  – Also provides an unprotected memory area which anyone can access, since considerable amount of data, for example input and output, belonging to the software should be accessed from other entities.
• For implementing security software (e.g. cryptographic protocol) with secure processor, the software programmer has to classify every data of the software (source code) according to protection requirement, and assigns appropriate memory area to every data.
• Difficulty of classification and assignment
  – Specialist knowledge of security is required.
  – There exist enormous amount of data (variables)
  – Once the confidential data is located on the unprotected area, it is hard to detect such an error!
  – Such vulnerabilities cannot be detected by a functional test and it becomes obvious only after the confidential data on the unprotected memory is found and cracked by adversaries!!!

DFITS (Data Flow Isolation Technology for Security)

1. What is DFITS?

• DFITS
  – Takes as input a source code,
  – Analyzes data flow of the source code focusing on cryptographic processing,
  – Classifies every data according to the protection requirement,
  – Generates executable program with appropriate memory assignment.
• By using DFITS system, we can develop security software without human error efficiently.
• DFITS assigns data (variable) to the memory appropriately.
• DFITS detects improper data transfer between protected and unprotected memory areas by finding protection attribute mismatch.

2. Basic data classification algorithm

(1) Protection attribute → classify data from the viewpoint of whether the data must have the confidentiality and the integrity
• Confidentiality: The data is accessible only to those authorized to have access
• Integrity: The data is consistent and correct

memory area: unprotected => confidentiality, integrity, usage example
   protected => confidentiality, integrity, usage example

 unprotected: exposed, input/output, public key
          verified, input/output, public key
          concealed, unverified, decrypted-message
          confidential, secret key

protected: exposed, input/output, public key
          verified, input/output, public key
          concealed, unverified, decrypted-message
          confidential, secret key

(2) Basic attribute inference algorithm

Constraint 1: Input/output of cryptographic functions
We can restrict the protection attributes of input/output for each cryptographic function.
Examples: Cryptographic functions
• Decryption function
  – Decryption key must have confidentiality and integrity
  – Plaintext must have confidentiality
  – The integrity of ciphertext is the same as that of the plaintext

Constraint 2: Isolation
In non-cryptographic functions, all variables within each processing must have the same attribute.

3. Additional algorithm for implicit integrity verification

• In order to analyze some existing cryptographic protocols (PGP, S/MIME, SSL, etc.), the analysis of a relation with two (or more) branch data flow is required.
• Representative case: implicit integrity verification
  – When the integrity of an output F(x) of the function F is verified, then the integrity of an input x is implicitly verified at the same time.
  – If, for the function F, no one can find two different inputs x1, x2 such that F(x1) = F(x2).
• In the additional algorithm, the integrity dependency relation described above is defined for each function
• Detect the variables whose integrity is implicitly verified, and modify the data flow

4. Prototype Implementation & Experimental Validation

• Prototype implementation of DFITS
  – Syntax analysis library Elsa[4] for syntax analysis
  – Class templates of C++ with suitable memory allocators
• Apply our prototype to a part of source code of OpenSSL
  – Remove the reuse of variables
  – Adaptation of source code to our prototype
• DFITS system appropriately outputs the protection attributes of at least 108 variables in the target source code.

Future and Related Works

• Improve DFITS in aspects of both theory (the classification algorithm) and implementation for applying DFITS system to much more cryptographic protocols.
• Comparison between DFITS and standard information flow analysis (IFA)[5]

<table>
<thead>
<tr>
<th></th>
<th>DFITS</th>
<th>Standard IFA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scope of analysis</strong></td>
<td>Confidentiality and integrity are considered from the viewpoint of whether cryptographic operation is applied to each data (Ex.) It is judged to be secure to publish a signature signed with a secret key, only when the message to be signed has the integrity and the secret key has both the confidentiality and the integrity.</td>
<td>Obtain pairs of arbitrary input and the output of the system, and observe their relations</td>
</tr>
<tr>
<td><strong>Assumptions</strong></td>
<td>Secure processor</td>
<td>Security processor constraints using the non-interference property, which prevents any flow of confidential data to public data (Ex.) It is judged to be insecure to publish a signature signed with a secret key.</td>
</tr>
</tbody>
</table>