ASSESSMENT OF DATA OBFUSCATION WITH RESIDUE NUMBER CODING

Biniam F. Demissie, Mariano Ceccato, Roberto Tiella
{demissie,ceccato,tiella}@fbk.eu
OUTLINE

• Problem Definition: Software Security
• State-of-the-art Defenses
• Obfuscation based on Residue Number Coding
• Empirical Assessment
INTRODUCTION

• Traditional computer security attack models were man-in-the-middle attacks
  • Message encryption
  • Policy & certificates
INTRODUCTION

• Different scenarios of man-at-the-end (MATE) attacks emerged
ATTACK MOTIVATIONS

• Malicious reverse engineering to understand an algorithm
ATTACK MOTIVATIONS

• Malicious tampering in order to alter functionality
There are different protection techniques

- Client-Server Splitting
- Code Signing
- Code Encryption
- Code Diversity

- Obfuscation
  - Control Obfuscation
  - Data Obfuscation
  - Layout Obfuscation
CHANGE ENCODING TRANSFORMATION (CET)

• It changes how values are represented in memory

• It might involve change in data-type

• It hides the original values in memory from attackers

```
short x;
x=10;
x=2x+40000
int x;
x=40020;
```
RESIDUE NUMBER CODING (RNC)

- An encoding based on RNC is CET that takes an integer value \( v \) and splits it into \( u > 0 \) components
  - i.e., \( E(v) = (\mu_1, \mu_2, \ldots, \mu_u) \)
EXAMPLE

• If $v=95$, $u=2$, $m_1 = 14$ and $m_2 = 15$, then
  - $\mu_1 = 95 \mod 14 = 11$
  - $\mu_2 = 95 \mod 15 = 5$

• Therefore, $E(v) = E(95) = (11, 5)$

• Hence, the value of $v$ is not seen in clear in memory
HOMOMORPHIC ENCODING

• An encoding function $E(\bullet)$ is homomorphic w.r.t $\bullet$ if, given $E(x)$ and $E(y)$, we can obtain $E(x \bullet y)$ without decoding $x$ and $y$

• That is, $E(x \bullet y) = E(x) \bullet' E(y)$

• Therefore, this allows certain computations on an encoded data
HOMOMORPHIC OPERATION IN RNC

• RNC is homomorphic with respect to addition (and hence, subtraction) and multiplication

i.e.,  
\[(\mu_1, \mu_2) +' (\partial_1, \partial_2) = (\mu_1 + \partial_1, \mu_2 + \partial_2)\]

\[(\mu_1, \mu_2) \times' (\partial_1, \partial_2) = (\mu_1 \times \partial_1, \mu_2 \times \partial_2)\]
OBFUSCATION TRANSFORMATION IMPLEMENTATION

• Data obfuscation based on RNC is applied as source-to-source transformation (for C/C++)

• The LLVM-Clang compiler framework was used to manipulate AST

• By traversing the AST, transformation is then applied by matching different patterns
```c
void main() {
    int a; // security sensitive
    int b = 20, c = 10; // sec inses.
    a = b + c;
}
```

**Encoded**

- `x: long long`
- `y: long long`
- `+Encoded(a: int): Encoded`
- `+DECODE(encoded: Encoded): int`

**Original code**

```c
void main() {
    int a; // security sensitive
    int b = 20, c = 10; // sec inses.
    a = b + c;
}
```

**Intermediate computation**

```plaintext
\[
\begin{align*}
    u &= 2, m_1 = 14, m_2 = 15 \\
    E(b) &= (\mu_1, \mu_2) = (1, 13) \\
    E(c) &= (\rho_1, \rho_2) = (11, 7) \\
    E(a) &= (\mu_1 + \rho_1, \mu_2 + \rho_2) \\
           &= (12, 20)
\end{align*}
\]
```

**Obfuscated code**

```c
void main() {
    Encoded a;
    int b = 20, c = 10;
    a = Encoded(b) + Encoded(c);
}
```
EMPIRICAL VALIDATION

• We want to address two research questions:
  
  • **RQ1**: what is the memory overhead for RNC based obfuscation?
  • **RQ2**: what is the runtime overhead for RNC based obfuscation?
METRICS & CASE STUDIES

✦ Metrics

• Memory, Time and Number of Obfuscated Variables

✦ Case studies

• License-check: a small routine program that takes license number as input and checks its validity

• MD5: an I/O and performance intensive program that computes file checksum
EXPERIMENTAL SETTING

License check
- 365 scenarios
- `main()` function is run 1000X

MD5
- 4 scenarios, 4 different inputs files
- 10 executions to avoid bias

In both cases, output of obfuscated code is compared with that of clear code
EXECUTION TIME (LICENSE-CHECK)

- Linear increase in execution time is observed
• Similar trend is observed
EXECUTION TIME (MD5)
MEMORY OVERHEAD (MD5)

- a) Spin
- c) Chrome
- d) Eclipse
OBSERVATION

• There is a linear increase in memory usage per number of obfuscated variables

• Runtime overhead depends on program complexity
  • Significant for small programs
  • Insignificant for I/O and performance intensive programs
CONCLUSION

MATE  Obfuscation  RNC, homomorphism

Data obfuscation based on RNC

It is lightweight
THANK YOU!

Questions?