Partitioning C program with dynamic taint analysis

Peng Xu

June 24, 2019
Dynamic taint analysis

1. Valgrind + taintgrind https://github.com/wmkhoo/taintgrind

2. Steps:
   - labeling the sensitive data
   - tracing the taint propagation
   - finding the functions and statements relative with labeled sensitive data

3. Example
   - tests/sign32.c
   - \texttt{TNT\_TAINT(&a, \text{sizeof}(a))};
   - \texttt{valgrind -tool=taintgrind tests/sign32}
   - \texttt{valgrind -tool=taintgrind tests/sign32 2>&1 \text{"python log2dot.py > sign32.dot"}}
   - \texttt{gcc -g}
Partitioning C program

1. Getting tainted information
   - Tainted files
   - Tainted functions
2. Pinpointing the tainted files and functions
3. Splitting program on the source code level
4. Generating splitted source code
   - Bare-metal system
   - SDK-based
5. Compiling and linking to binary
2. Taint Analyzer
   - Method-level tainting (TZ-M)
   - Block-level tainting (TZ-B)
   - Line-level tainting (TZ-L)
TZSlicer


2. Taint Analyzer
   - Method-level tainting (TZ-M)
   - Block-level tainting (TZ-B)
   - Line-level tainting (TZ-L)

3. Program Slicer:
   - Conducts the slicing
   - Generates the two program slices based on the results obtained from the Taint Analyzer
   - should be functionally equivalent
   - has world switching code
     - Shared memory access: shared memory
     - World switching: secure monitor
TZSlicer


2. Taint Analyzer
   - Method-level tainting (TZ-M)
   - Block-level tainting (TZ-B)
   - Line-level tainting (TZ-L)

3. Program Slicer:
   - Conducts the slicing
   - Generates the two program slices based on the results obtained from the Taint Analyzer
   - should be functionally equivalent
   - has world switching code
     - Shared memory access: shared memory
     - World switching: secure monitor

4. Slicer optimization
   - Resource optimization: TZ-M, TZ-B and TZ-L slicing
   - Communication optimization: Loop-unrolling and variable rename
Glamdring

1. 

2. Code annotation: sensitive data labeling
   - Sensitive source
   - Sensitive sink
Glamdring


2. Code annotation: sensitive data labeling
   ▶ Sensitive source
   ▶ Sensitive sink

3. Code analysis
   ▶ Static dataflow analysis for confidentiality
   ▶ Static backward slicing for integrity
Glamdring


2. Code annotation: sensitive data labeling
   ▶ Sensitive source
   ▶ Sensitive sink

3. Code analysis
   ▶ Static dataflow analysis for confidentiality
   ▶ Static backward slicing for integrity

4. Code partitioning - sensitive functions
   ▶ Security-sensitive functions
   ▶ Memory allocations
   ▶ Global variables
Glamdring


2. Code annotation: sensitive data labeling
   ▶ Sensitive source
   ▶ Sensitive sink

3. Code analysis
   ▶ Static dataflow analysis for confidentiality
   ▶ Static backward slicing for integrity

4. Code partitioning - sensitive functions
   ▶ Security-sensitive functions
   ▶ Memory allocations
   ▶ Global variables

5. Code generation
   ▶ Source-code level transformation
   ▶ Moving function definitions into the enclave
   ▶ Generating ecalls and ocalls
sensitive data

encryption engine

decryption engine

logical operation
logicProcessing(sensitiveData){...}

decryption engine

encryption engine

encryption engine

syncx Server

secure world

enclave interface

decryption engine

encryption engine

normal world

decryption engine

encryption engine

Client

sensitive data

encryption engine

decryption engine

Client

sensitive data

encryption engine

decryption engine

Client

sensitive data

encryption engine

decryption engine

normal world

decryption engine

encryption engine

secure world

logical operation
logicProcessing(sensitiveData){...;
SubChild1 * subChild1 = new SubChild1();
subChild1 -> show(sensitiveData);
sensitiveData = fake_sensitiveData(...);
}
main.c
void main()
{  ....
   TNT_TAINT(key,sizeof(key));
   ....
}
main.cpp
int main(int argc, char *argv[])
{  ....
   char *dataptr = data;
   TNT_TAINT(&dataptr,sizeof(data));
   subchild1->show(dataptr);
   ....
   return 1;
}

gcc main.c -o c_prog
g++ main.cpp -o cpp_prog

Taint Annotation

Dynamic Taint Analyzer

Function Level
S: 1. KeyExpansion
2. getSBoxValue
3. main
N: FFT, _encrypt, _decrypt, test_encrypt_cbc

Class Level
S: 1. SubChild1::show(char*)
2. output(char*)
3. main
N: Base::show, Child::show

Variables List

Taint Analysis

Partitioning Policy Generation

Sliced Code Generator

Enclave
secure world

Normal World

SGX SDK

Sliced Code Generator

OCALL
ECALL
// Classes declaration
class Base{
public:
    int mBase;
    virtual void show(int);
    virtual void show(char * a);
};
class Child1 : public Base{
public:
    int mChild1;
    virtual void show(int);
};
class Child2 : public Base{
};
class SubChild1 : virtual public Child1{
public:
    int mSubChild1;
    virtual void show(int);
    virtual void show(char *a);
};

// Methods definition
void Base::show(int){}
void SubChild1::show(char *){
    decrypt(ch);
    logicalFunc(ch);
    encrypt(ch);
}

int main(int argc, char * argv[]){
    char *dataptr = data;
    TNT_TAINT(&dataptr,sizeof(data));
    subchild1->show(dataptr);
    return 1;
}
Question?

Questions?