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
   - *TNT_TAINT(&a, sizeof(a));*
   - *valgrind –tool=taintgrind tests/sign32*
   - *valgrind –tool=taintgrind tests/sign32 2>&1 — python log2dot.py > sign32.dot*
   - *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

TZSlicer

1. https://ieeexplore.ieee.org/document/8383886
2. Taint Analyzer
   - Method-level tainting (TZ-M)
   - Block-level tainting (TZ-B)
   - Line-level tainting (TZ-L)

TZSlicer

1. https://ieeexplore.ieee.org/document/8383886
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

1. https://ieeexplore.ieee.org/document/8383886
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. https://www.usenix.org/system/files/conference/atc17/atc17-lind.pdf
2. Code annotation: sensitive data labeling
   - Sensitive source
   - Sensitive sink

Glamdring

1. https://www.usenix.org/system/files/conference/atc17/atc17-lind.pdf
2. Code annotation: sensitive data labeling
   - Sensitive source
   - Sensitive sink
3. Code analysis
   - Static dataflow analysis for confidentiality
   - Static backward slicing for integrity

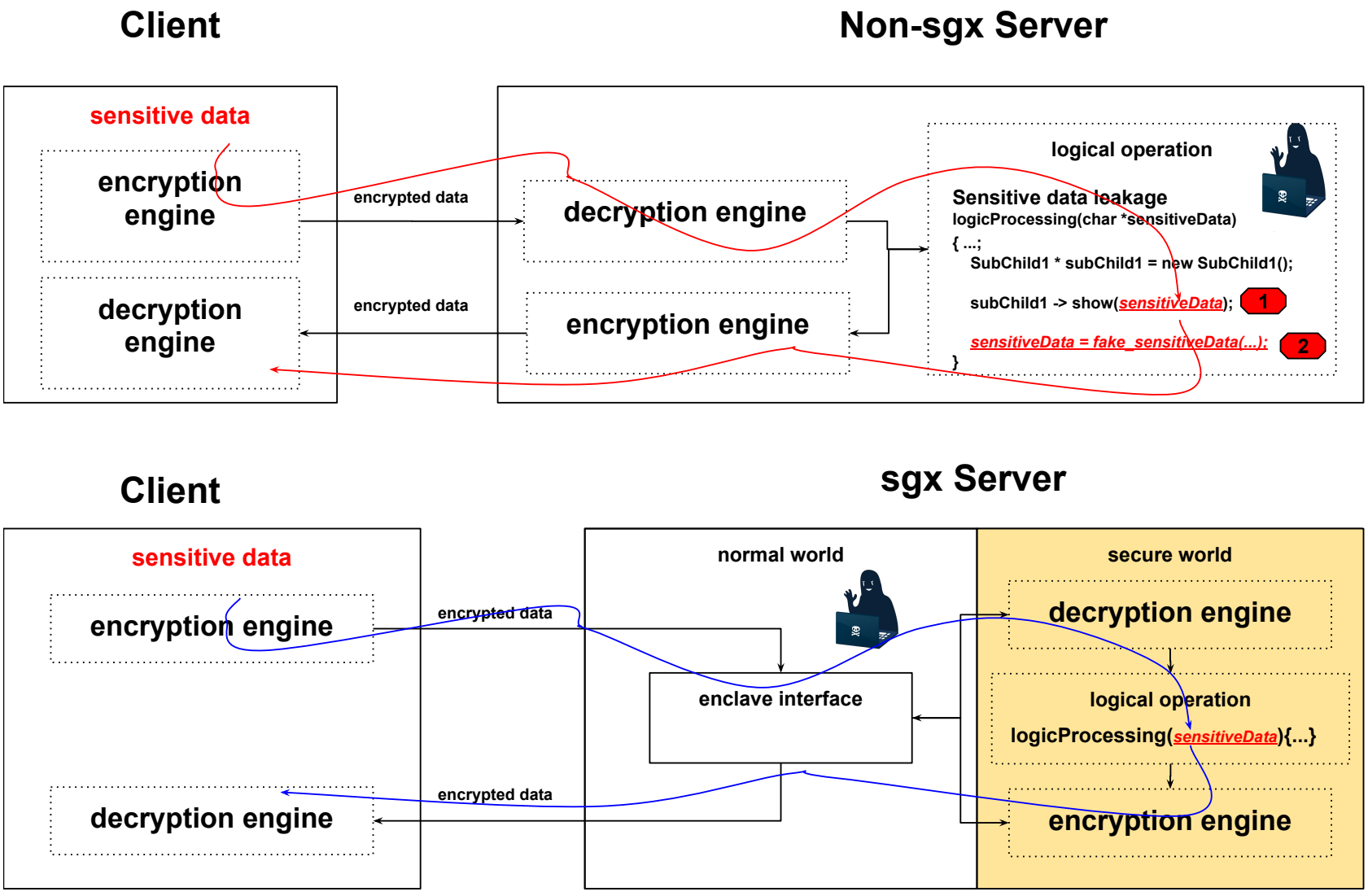
Glamdring

1. https://www.usenix.org/system/files/conference/atc17/atc17-lind.pdf
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

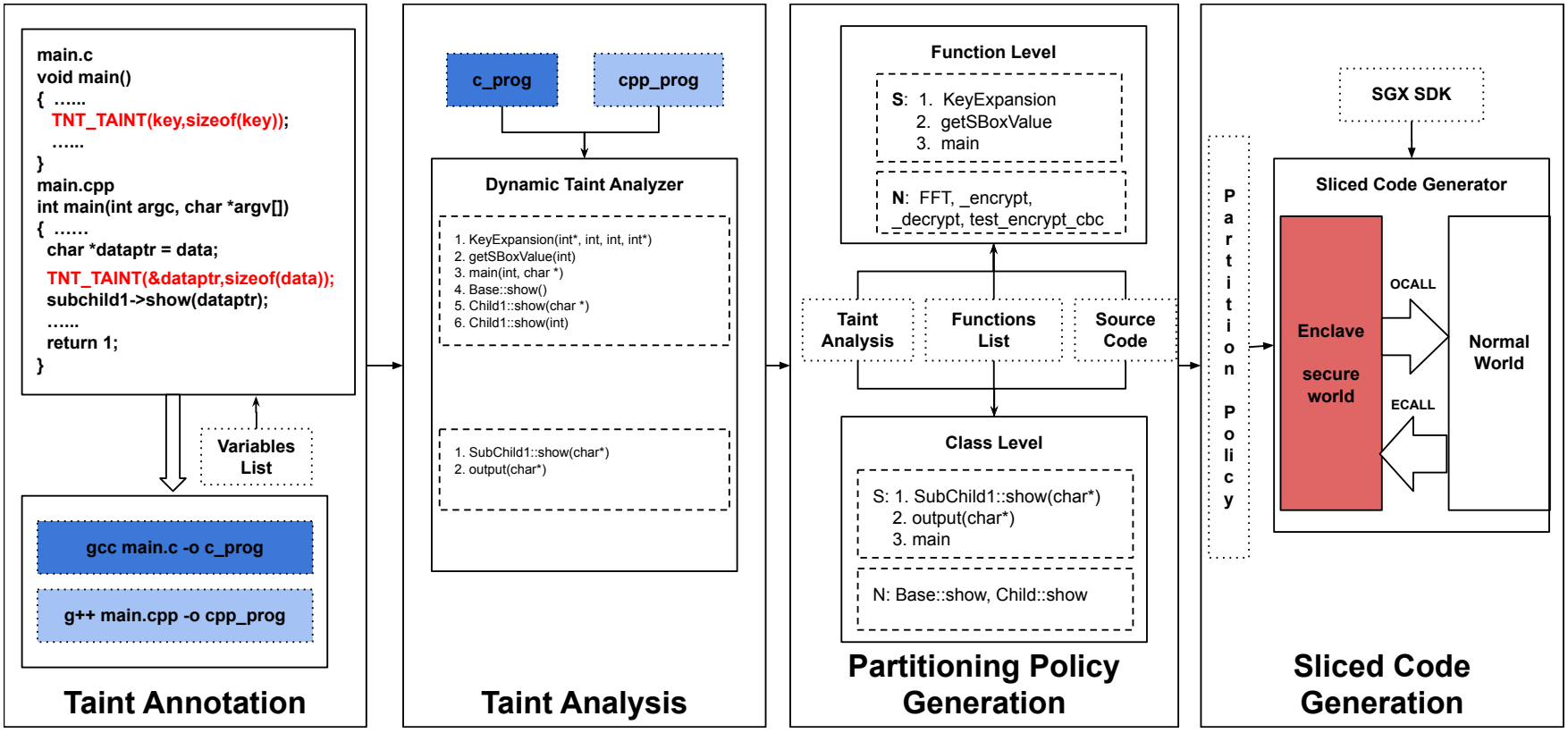
Glamdring

1. https://www.usenix.org/system/files/conference/atc17/atc17-lind.pdf
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

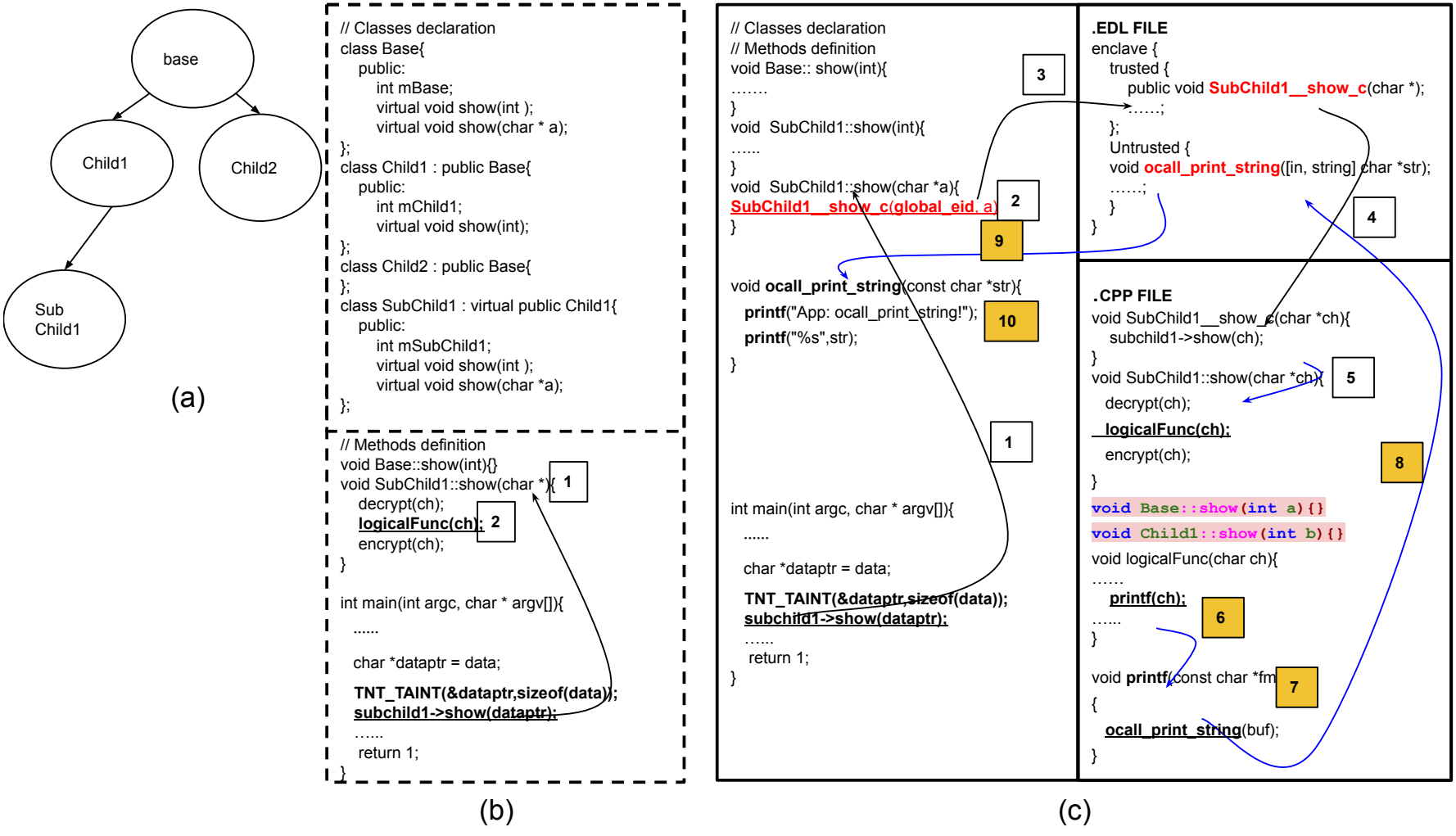
AppSlicer


Client
Non-sgx Server
sensitive data
encryption engine
decryption engine
encrypted data
encrypted data
decryption engine
encryption engine
logical operation
Sensitive data leakage
logicProcessing(char *sensitiveData)
{ ...;
SubChild1 * subChild1 = new SubChild1();
subChild1 -> show(sensitiveData);
1
sensitiveData = fake_sensitiveData(...);
2
}
Client
sgx Server
sensitive data
encryption engine
decryption engine
encrypted data
encrypted data
normal world
enclave interface
secure world
decryption engine
logical operation
logicProcessing(sensitiveData){...}
encryption engine

AppSlicer


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;
}
Variables List
gcc main.c -o c_prog
g++ main.cpp -o cpp_prog
Taint Annotation
c_prog
cpp_prog
Dynamic Taint Analyzer
1. KeyExpansion(int*, int, int, int*)
2. getSBoxValue(int)
3. main(int, char *)
4. Base::show()
5. Child1::show(char *)
6. Child1::show(int)
1. SubChild1::show(char*)
2. output(char*)
Taint Analysis
Function Level
S: 1. KeyExpansion
2. getSBoxValue
3. main
N: FFT, _encrypt, _decrypt, test_encrypt_cbc
Taint Analysis
Functions List
Source Code
Class Level
S: 1. SubChild1::show(char*)
2. output(char*)
3. main
N: Base::show, Child::show
Partitioning Policy Generation
SGX SDK
Partition Policy
Sliced Code Generator
Enclave
secure world
OCALL
ECALL
Normal World
Sliced Code Generation

AppSlicer


base
Child1
Child2
Sub Child1
(a)
// 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;
}
1
2
(b)
Normal World
Secure World
// Classes declaration
// Methods definition
void Base:: show(int){
.....
}
void SubChild1::show(int){
.....
}
void SubChild1::show(char *a){
SubChild1__show_c(global_eid, a);
}
void ocall_print_string(const char *str){
printf("App: ocall_print_string!");
printf("%s",str);
}
int main(int argc, char * argv[]){
.....
char *dataptr = data;
TNT_TAINT(&dataptr,sizeof(data));
subchild1->show(dataptr);
.....
return 1;
}
.EDL FILE
enclave {
trusted {
public void SubChild1__show_c(char *);
.....;
};
Untrusted {
void ocall_print_string([in, string] char *str);
.....;
}
}
.CPP FILE
void SubChild1__show_c(char *ch){
subchild1->show(ch);
}
void SubChild1::show(char *ch){
decrypt(ch);
logicalFunc(ch);
encrypt(ch);
}
void Base::show(int a){}
void Child1::show(int b){}
void logicalFunc(char ch){
.....
printf(ch);
.....
}
void printf(const char *fm
{
ocall_print_string(buf);
}
1
2
3
4
5
6
7
8
9
10
(c)

Question?

Questions?