Hiding program slices for software security

X. Zhang and R. Gupta
University of Arizona

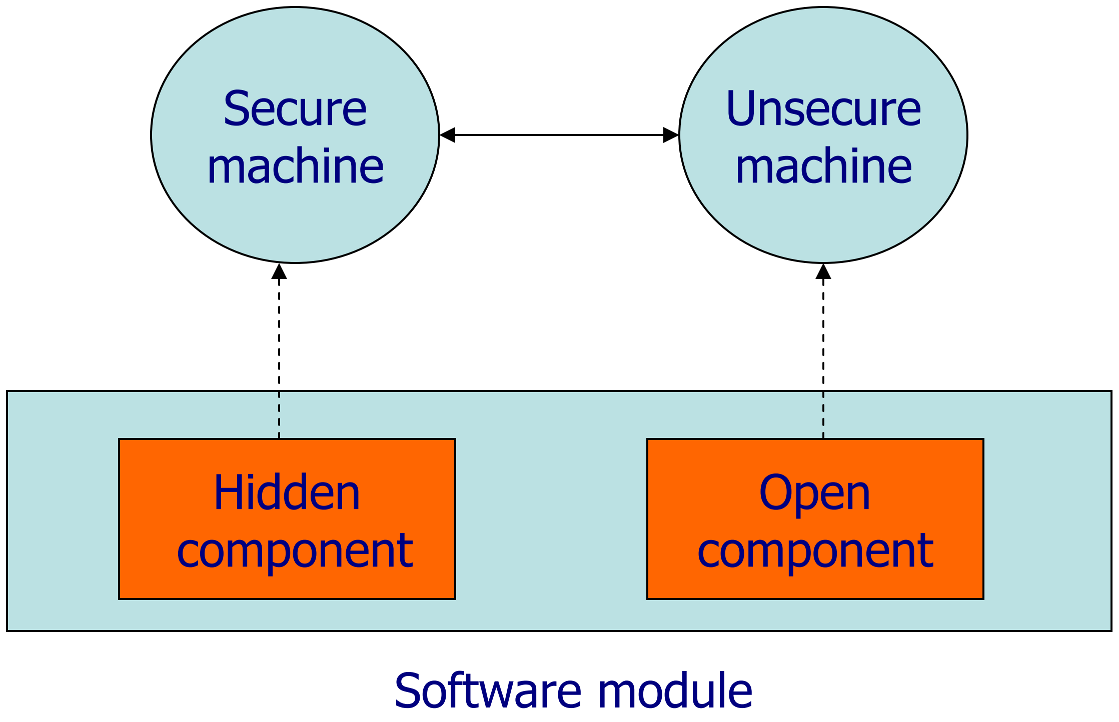
Software Piracy

- Software protection technique to prevent software piracy based on program slicing
- Prevent the malicious user to gain a working copy of the software that can be distributed for illegal use
- Does not prevent tampering

Idea: Software Splitting

- Split software modules into *open* and *hidden* components:
  - *Open components* installed and executed on unsecure machine
  - *Hidden components* installed and executed on secure machine
- Open components can be stolen but they are incomplete (they only provide a subset of the application functionality)
- Similar to server side execution

Idea: Software Splitting



Challenges

- Resilience

  Deriving the hidden components by observing the code of the open components and their run-time interactions with the hidden components requires a great deal of effort

- Cost

  Limit the communication between hidden and open components

Splitting Transformation

- (S,C) program runtime state and code

Hidden component

State	Code
$S' + s$	$C' + c$

Open component

State	Code
$S - S' + s$	$C - C' + c$

- (s,c) additional variables and new code implementing the interaction between components

Hiding Modules

- Select one or more complete modules and treat them as hidden components does not work because the attacker could guess the functionality of the module
- Assuming that the attacker cannot guess the functionality of the module, we still need to find suitable module for hiding
- These modules should be self-contained but self-contained modules are not very common

Hiding Module Slices

- Construction of hidden components out of program slices such that their behaviour cannot be easily understood

- A program slice is composed by:
  - Variables
  - Expressions and assignments
  - Control flow

Variables

- Consider a function f and a subset of hidden variables of f:
  - Hidden components Hf that maintains the hidden variables
  - Open component Of

- Interaction between Hf and Of:
  - When Of computes a new value for a hidden variable v the new value is sent to Hf to update it
  - When Of needs to use v it recieves the current value from Hf

- All the references to hidden variables in Of are replaced by a single variable v in Of

- Dynamic analysis can recover the hidden variables

Expressions and Assignments

- Some statements that affect the values of hidden variables are moved to the hidden component
- All the statements that belong to the *forward-data slices* constructed by following data dependence edges originating at definitions of hidden variables
- An hidden variable may cause additional variables to be hidden (or partially hidden) in Hf
- More difficult to estabilish relations between the values that are exchanged between Hf and Of:
  - we do not know how many variables are hidden and
  - the form of expressions that matain them

Control Flow

- Move control ancestors of selected statements that belong to forward data slices of hidden variables
- Control ancestors are hidden if doing so will simultaneously introduce a control flow construct in Hf and remove or alter the control flow in Of
- Moving control flow in Hf makes the task of recovering hidden components more difficult

Function Selection

- The overall cost depends on the number of functions that are selected for splitting
- Contruct the call graph, identify a cut and split the functions that are part of the cut
  - Avoid functions that are called from inside a loop
  - No functions calls made by f are hidden in Hf
  - Only scalar variables local to f are considered as candidate hidden variables

Function Splitting

- Select a function *f* and a local variable *v* for splitting
- *Hf* is given by fragments of code (statements) of *f* identified by an unique label ID
- In *Of* there are calls to *Hf* in the points where the statements have been removed: Hf([needed Of values],ID)

Function Splitting

- Step 1: construct Slice(f,v) starting from the statements defining *v*
- Step 2: examine the statements in *f* and Slice(f,v) to determine the set of hidden variables
- Step 3: split each statement *lhs* Ã *rhs* in Slice(f,v) between Hf and Of
  - Both lhs and rhs in Hf
  - Only lhs in Hf, because rhs cannot be placed in Hf (function call)
  - Only rhs in Hf, because variable lhs cannot be placed in Hf (array)
  - None

Function Splitting

- Step 4: examine the statements that are not in Slice(f,v) but that contain a reference/use/definition to a partially transferred variable
  - *x* Ã *rhs* and *x* is partially hidden: *rhs* is evaluated on Of and the result is sent to Hf in order to update the value of variable *x*
  - *lhs* Ã *rhs* and *rhs* refers to *x*: a call to Hf preceeds this statement in Of in order to obtain the value of *x*

Example

```
function f(…)
int a,b,c
int …
…
a Ã 3x + y
…
b Ã a + w
…
A[b-1] Ã …
…
if (y>10) then
c Ã a*x +w
else
C Ã 2x +w
endif
```

```
function Of()
int c,t
int…
 …
t Ã Hf([x,y],l1)
 …
t Ã Hf([w],l2)
 …
A[Hf([],l3)] Ã …
…
t Ã Hf([y,x,w],l4)
  if (t == 1) then
  c Ã 2x +w
  endif
```

```
function Hf(int[],id)
  static  int a,b,c
  switch id
     …
 l1: a Ã 3t[0] + t[1]
     return(any)
 l2: b Ã a + t[0]
     return(any)
 l3: return(b-1)
  l4: if (t[0] > 10) then
  c Ã a*t[1] + t[2]
  endif
   return((t[0]>10)?:0:1)
```

Complexity of Hf

- leaked value (lv): *lv = f(observable values)*
- Arithmetic complexity of f :

  AC(f,P) = <Type, Inputs, Degree>
- Control flow complexity of f:

  CC(f) = <Paths, Predicates, Flow>

Software Splitting

- The complexity of the hidden components guarantees that it is difficult for an attacker to recover the hidden component
- Algorithm for measuring arithmetic and flow complexity of lv = f(P, observable values used). This algorithm helps in choosing between different splitting options
- Run time overhead 4% - 58%