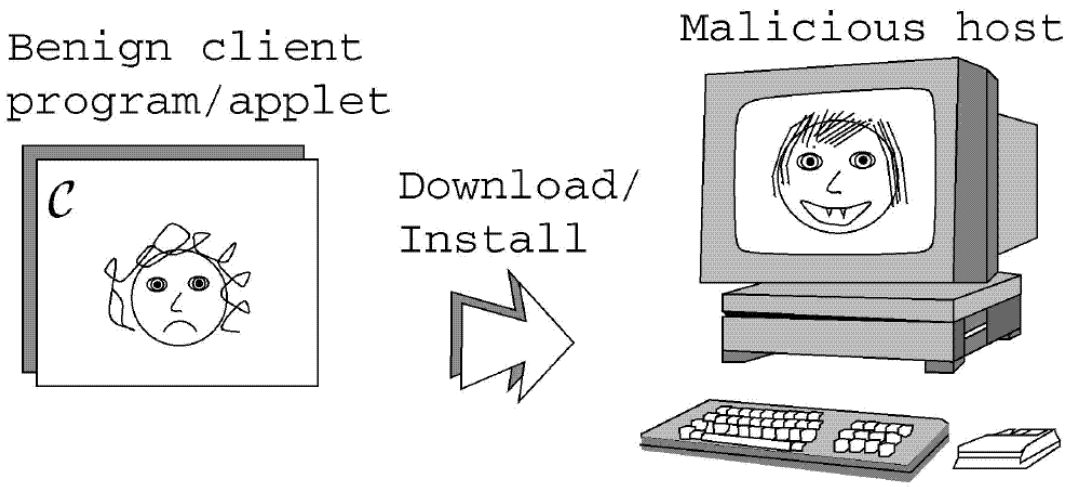
Software Security: Obfuscation

Shuai Wang

香 港 科 技 大 學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

Software Protection

- The problem of mobile code and “malicious” hosts.



Motivation of Software Protection

- Software contains code, which is highly informative..
  - MSIL for .NET (C#)
  - Bytecode for Java/Javascript/Python
  - Binary code for C/C++
- But code can be exploited
  - Non-malicious RE: for testing, integration, extending, …
  - Malicious RE: for security attacking, piracy, …
- Well, malware authors also want to evade malware clustering/similarity analysis…
  - In the sense, “malicious” host becomes the victim users…

Existing Protection Techniques

- Hardware Isolation
  - Not viable. Retrofit hardware needed.
- Server-side execution
  - High bandwidth always-on requirement.
  - Additional problems of network security – authentication need to be taken care of.
- Encryption
  - Chicken-egg problem … the decryption routine is visible.

Software Obfuscation

- Generally speaking, making protected software looks dissimilar to its original version.
  - From a very holistic perspective, determining the "similarity" of two pieces of code is the building block of many software security analysis

Software obfuscation can be very effective, at least visually

- The problem of software piracy and malicious reverse engineering …
  - Reuse (part) of your code, your algorithm, your homework assignments…

```
function hello(name) {
 console.log('Hello, ' + name);
}
hello('New user');
```

Yes, they are actually the same..

```
eval(function(p,a,c,k,e,d){e=function(c){return
c};if(!''.replace(/^/,String)){while(c--){d=k||c}k=[function(e)
{return d[e]}];e=function(){return'\\w+'};c=1};while(c--){if(k)
{p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k)}}return p}('3 0(1)
{2.4(\'5, \'+1)}0(\'7
6\');',8,8,'hello|name|console|function|log|Hello|user|New'.split('|
'),0,{}))
```

Software obfuscation

- Can never completely protect an application from malicious reverse engineering.
  - Given sufficient time and resources, an adversary can reverse engineer any obfuscated code. → exhaustive search
  - But use transforms such that the resources required for undoing them are too expensive for attackers.

```
eval(function(p,a,c,k,e,d){e=function(c){return
c};if(!''.replace(/^/,String)){while(c--){d=k||c}k=[function(e)
{return d[e]}];e=function(){return'\\w+'};c=1};while(c--){if(k)
{p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k)}}return p}('3 0(1)
{2.4(\'5, \'+1)}0(\'7
6\');',8,8,'hello|name|console|function|log|Hello|user|New'.split('|
'),0,{}))
```

Potential application domains

- Good ones …
  - Obscure program logic.
  - Protect ownership information
- Bad ones …
  - Development of malware or code that contains obfuscated malicious payload.
  - Code plagiarism
  - Algorithm plagiarism
  - Homework plagiarism..?
- Besides software, obfuscation is also popular in protecting information leakage in system/networking environment
  - Identity disclosure
  - Attack on Trusted Computing Base (TCB)..
  - …

Software Obfuscations

- Layout obfuscation
- Data obfuscation
- Control-flow obfuscation

Layout Obfuscation

- Changes or removes useful information from the code without affecting real instructions:
  - comment stripping, identifier renaming.
- Used in commercial obfuscators like DashO for Java and Dotfuscator for MSIL
  - Very lightweight methods…
  - You will be surprised if you see popular Android/iOS apps are not “layout obfuscated”

Layout Obfuscation

```
public class test1{
    private int term1;
    private int term2;
    private boolean areRelativelyPrime;

    public test1(int term1, int term2){
        this.term1=term1;
        this.term2=term2;
        areRelativelyPrime=areRelativelyPrime();
    }

    public static int gcd(int term1, int term2){
        int remainder;
        remainder=term1%term2;
        if (remainder==0){
            return term2;
        }
        else{
            return gcd(term2, remainder);
        }
    }

    private boolean areRelativelyPrime(){
        if (gcd(term1, term2)==1){
            return true;
        }
        else{
            return false;
        }
    }

    public static void main(String args[]) {
      test1 a=new test1(12, 19);
   }
}
```

What is changed?

```
public class a{
    private int a;
    private int b;
    private boolean c;

    public a(int a, int b){
        this.a=a;
        this.b=b;
        c=c();
    }

    public static int b(int a, int b){
        int c;
        c=a%b;
        if (c==0){
            return b;
        }
        else{
            return b(b, c);
        }
    }

    private boolean c(){
        if (b(a, b)==1)\{
            return true;
        }
        else{
            return false;
        }
    }

    public static void main(String args[]) {
      a b=new a(12, 19);
    }
}
```

Layout Obfuscation

```
@interface Person: NSObject
@property NSString *name;
@property int age;
@property NSString *addr;
@end
```

```
@interface AlJi09: NSObject
@property NSString *KJihad;
@property int z9kmV;
@property NSString *Nm23d;
@end
```

Identifier renaming in real-world iOS apps

Layout Obfuscation

Original Source Code Before Rename Obfuscation

```
private void
CalculatePayroll(SpecialList
employeeGroup) {
    while(employeeGroup.HasMore()) {
        employee =
employeeGroup.GetNext(true);
        employee.UpdateSalary();
        DistributeCheck(employee);
    }
}
```

Reverse-Engineered Source Code After Rename Obfuscation

```
private void a(a b) {
    while (b.a()) {
        a = b.a(true);
        a.a();
        a(a);
    }
}
```

Obfuscation in Android apps

Layout Obfuscation

What happened to the function name?

Data Obfuscations

- Variable splitting and merging and encoding
  - Arrays can be *split* into several sub-arrays
  - two or more arrays can be *merged* into one bigger array
  - *Arrays are folded* so as to increase the number of dimensions.
  - *Arrays are flattened* to decrease the number of dimensions.
  - Data are encrypted..
- Also very commonly used in mobile apps…

```
const char *str1 = "A␣plain␣string";
```

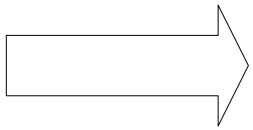
```
// string xor masked by 0xab
const char *str1 =
  "\xea\x8b\xdb\xc7\xca\xc2\xc5\x8b"
  "\xd8\xdf\xd9\xc2\xc5\xcc\x85";
void decode(const char *s, char *d)
{
  while(*s) *d++ = *s++ ^ 0xab;
  *d = 0;
}
```

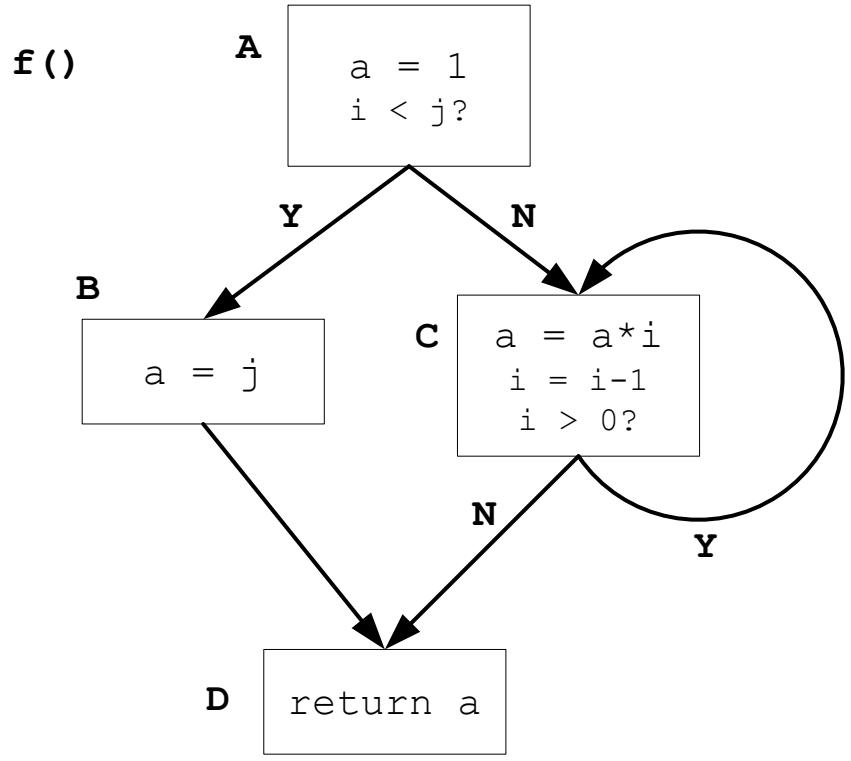
String encoding in real-world iOS apps

Control-flow Obfuscations

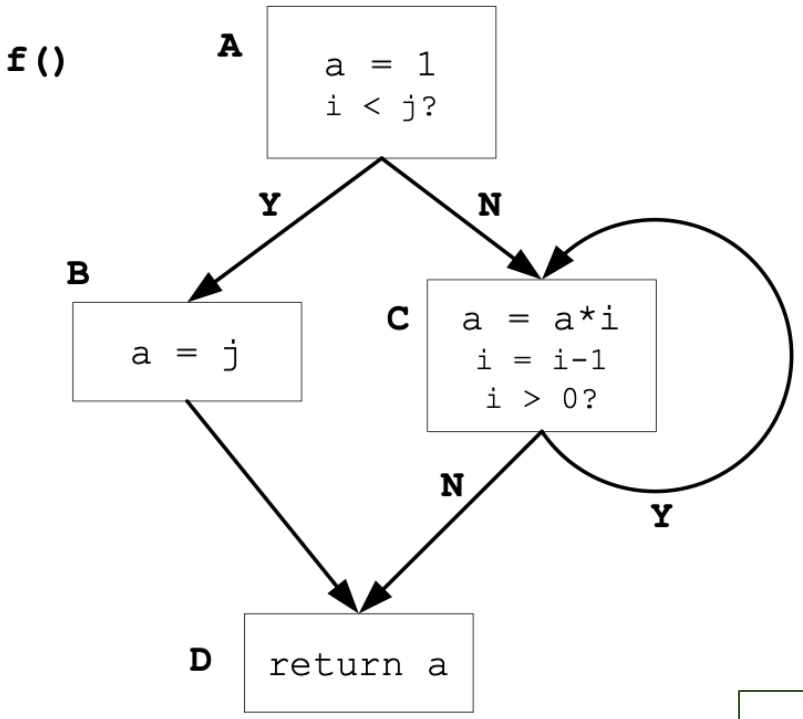
Background information: program control flow graph

```
int f(int i, int j)
{
      int a = 1;
      if (i < j) {
            a = j;
      }
      else
            do {
                  a *= i--;
            } while (i > 0);
      return a;
}
```





Motivation of Control-flow Obfuscations



renaming and data obfuscation does not change the control structure of the program.

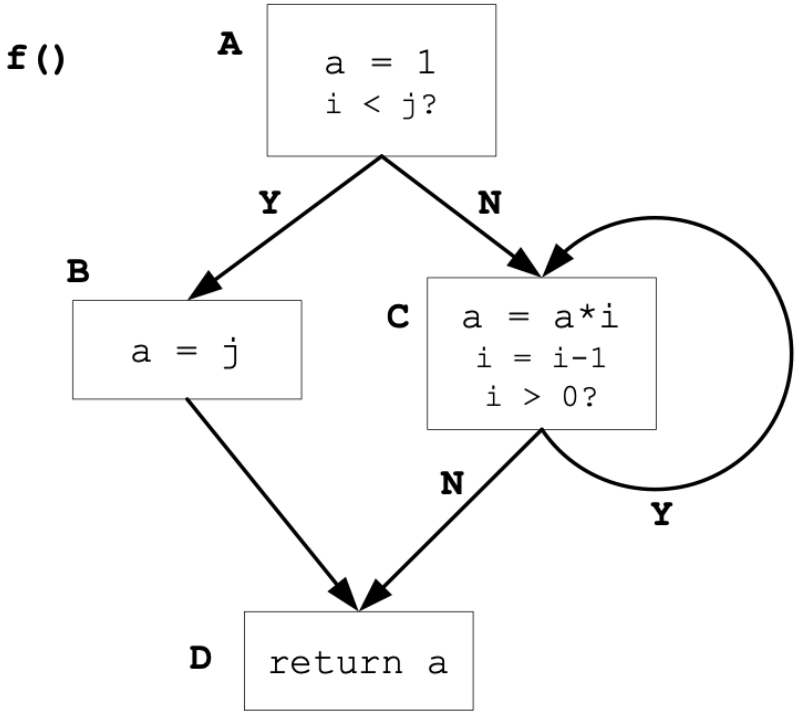
- The structure is preserved.

- Adversarial similarity analysis
- Code reuse attacks
- …

Can still leverage control structure as code signature

Control-flow Obfuscations

Control-flow obfuscations mutate program control structures in terms of different granularities.



Control-flow structure of function **f()**

Class	Methods
Instruction Level	instruction replace instruction insert
Basic Block Level	basic block reorder basic block merge basic block split opaque predict insert control flow flatten branch function insert
Function Level	function reorder function inline

Typical control-flow obfuscations

Let's talk about representative cases..

Instruction Insertion (Garbage Code Insertion)

- Insertion meaningless instructions or sequences of instructions
  - Single instruction: nop
  - A sequence of instructions: add → sub; xor → xor
- Very popular method
  - Easy to implement
  - Can easily break known patterns/signatures

Instruction Replacement/Substitution

before	after

`movsb`	`push eax`
	`mov al, [esi]`
	`inc esi`
	`mov [edi], al`
	`inc edi`
	`pop eax`

`mov eax, 0`	`xor eax, eax`
`add eax, 1`	`not eax` `neg eax`

Functionality is still the same!!

An extreme case of Instruction Substitution

All “mov” is also feasible…

```
<is_prime>:
  push   ebp
  mov    ebp,esp
  sub    esp,0x10
  cmp    DWORD PTR [ebp+0x8],0x1
  jne    8048490 <is_prime+0x13>
  mov    eax,0x0
  jmp    80484cf <is_prime+0x52>
  cmp    DWORD PTR [ebp+0x8],0x2
  jne    804849d <is_prime+0x20>
  mov    eax,0x1
  jmp    80484cf <is_prime+0x52>
  mov    DWORD PTR [ebp-0x4],0x2
  jmp    80484be <is_prime+0x41>
  mov    eax,DWORD PTR [ebp+0x8]
  cdq
  idiv   DWORD PTR [ebp-0x4]
  mov    eax,edx
  test   eax,eax
  jne    80484ba <is_prime+0x3d>
  mov    eax,0x0
  jmp    80484cf <is_prime+0x52>
  add    DWORD PTR [ebp-0x4],0x1
  mov    eax,DWORD PTR [ebp-0x4]
  imul   eax,DWORD PTR [ebp-0x4]
  cmp    eax,DWORD PTR [ebp+0x8]
  jle    80484a6 <is_prime+0x29>
  mov    eax,0x1
  leave
  ret
```

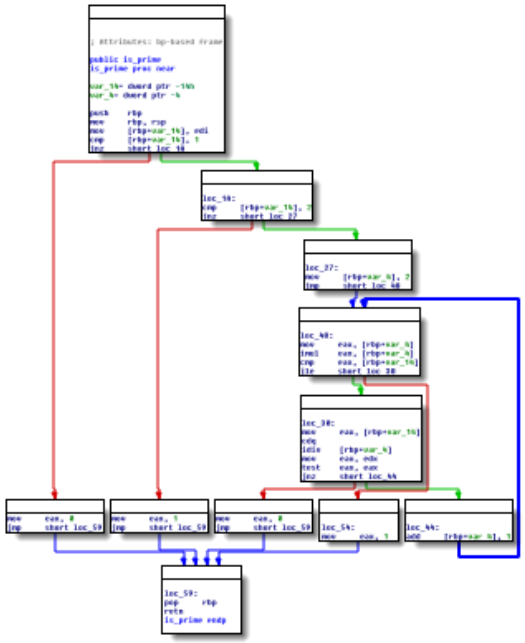
Original code

After obfuscation

M/o/Vfuscator compiles C/C++ programs into "mov" instructions, and only "mov" instructions.

An extreme case of Instruction Substitution

All “mov” is also feasible…



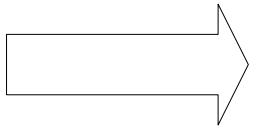
Original control flow structure

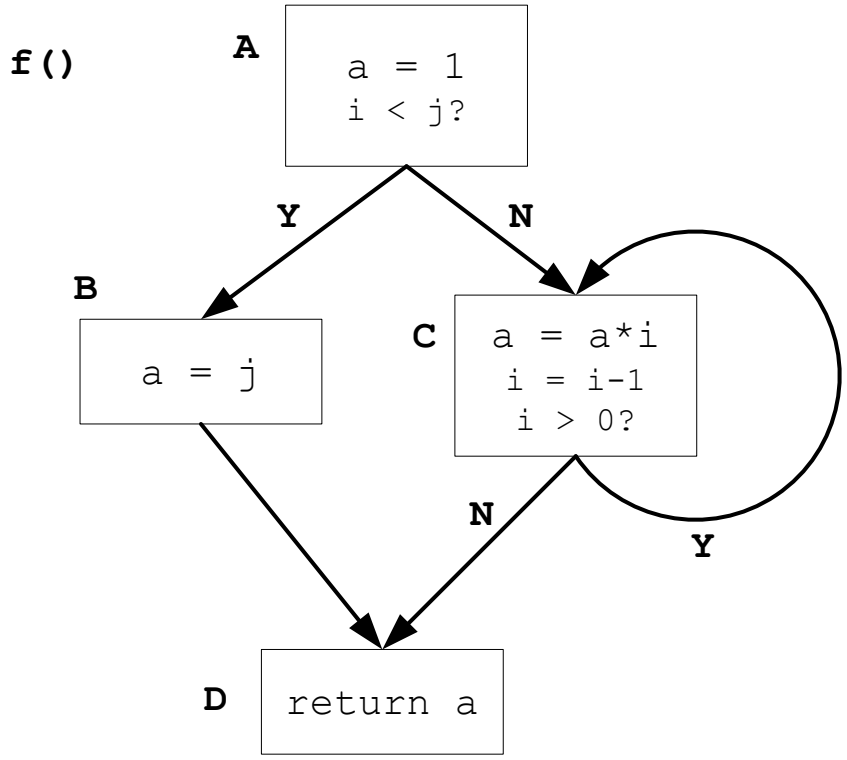
Control structure after obfuscation

M/o/Vfuscator compiles C/C++ programs into "mov" instructions, and only "mov" instructions.

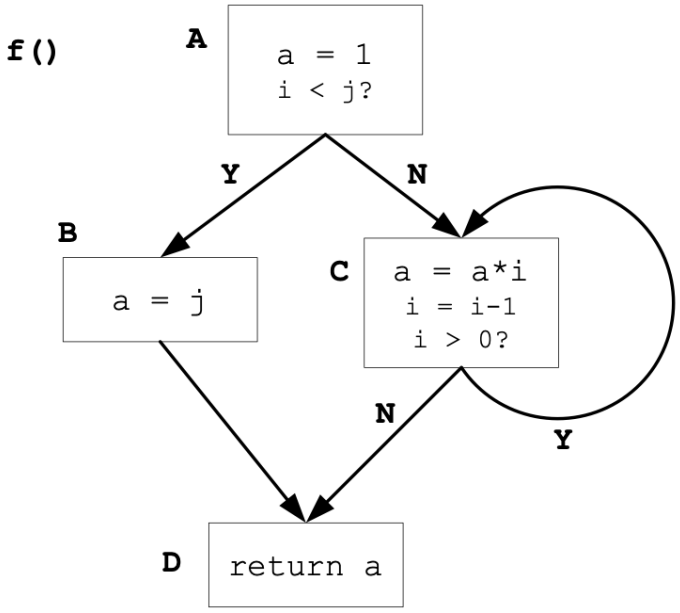
Control Flow Flattening

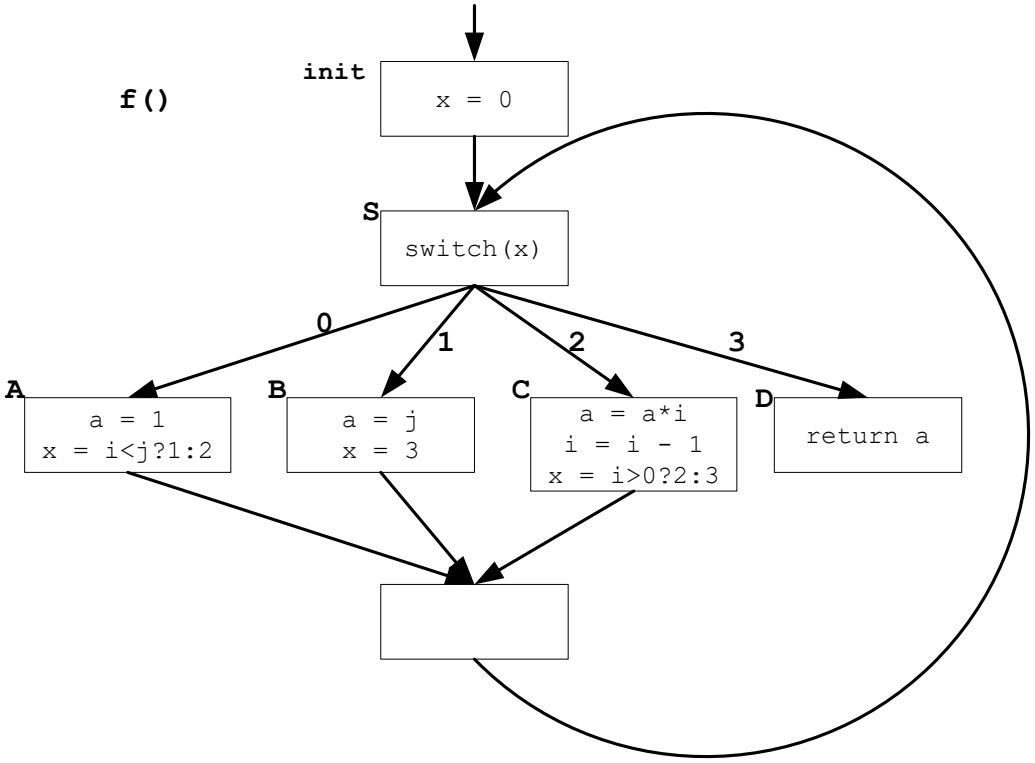
```
int f(int i, int j)
{
      int a = 1;
      if (i < j) {
            a = j;
      }
      else
            do {
                  a *= i--;
            } while (i > 0);
      return a;
}
```





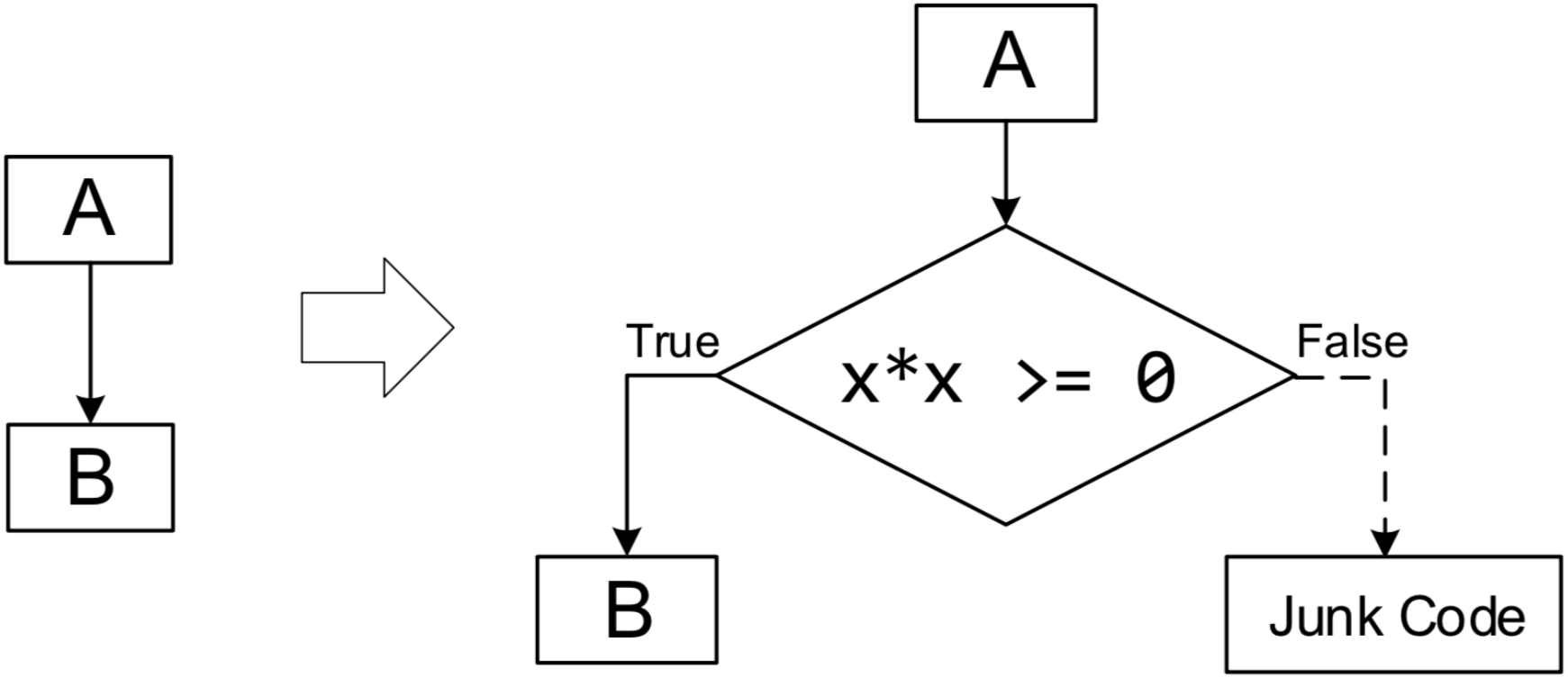
Control Flow Flattening





Opaque Predicates

predicate: conditional expression



What property an “opaque predicate” must satisfy?

Opaque Predicates

- An opaque predicate (Φ):
  - Value is known to the obfuscator
  - Value is easy to compute during runtime
  - Value difficult for the adversary to deduce (by statically analysing the code) → thus called *opaque*

- The *opacity* property of predicates determines the resilience of control-flow transformations, i.e.

  ↑ opaque a predicate ⇒ ↑ difficulty in determining its outcome.

Opaque Predicates

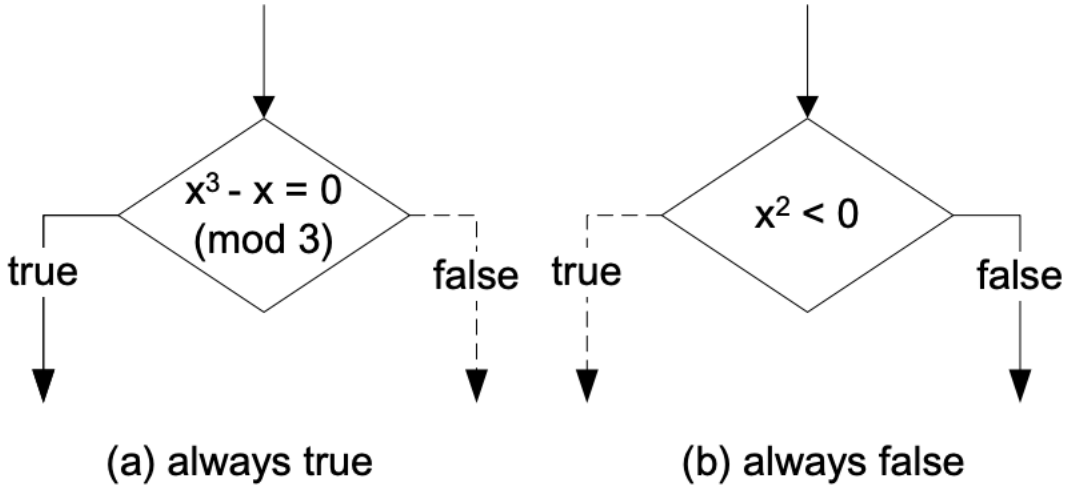
- A simple example
- Any math identity will work

```
if (x*x + y*y >= 2*x*y) …
```

  - …is always true, but not so obvious
- In assembly, this would be even less obvious

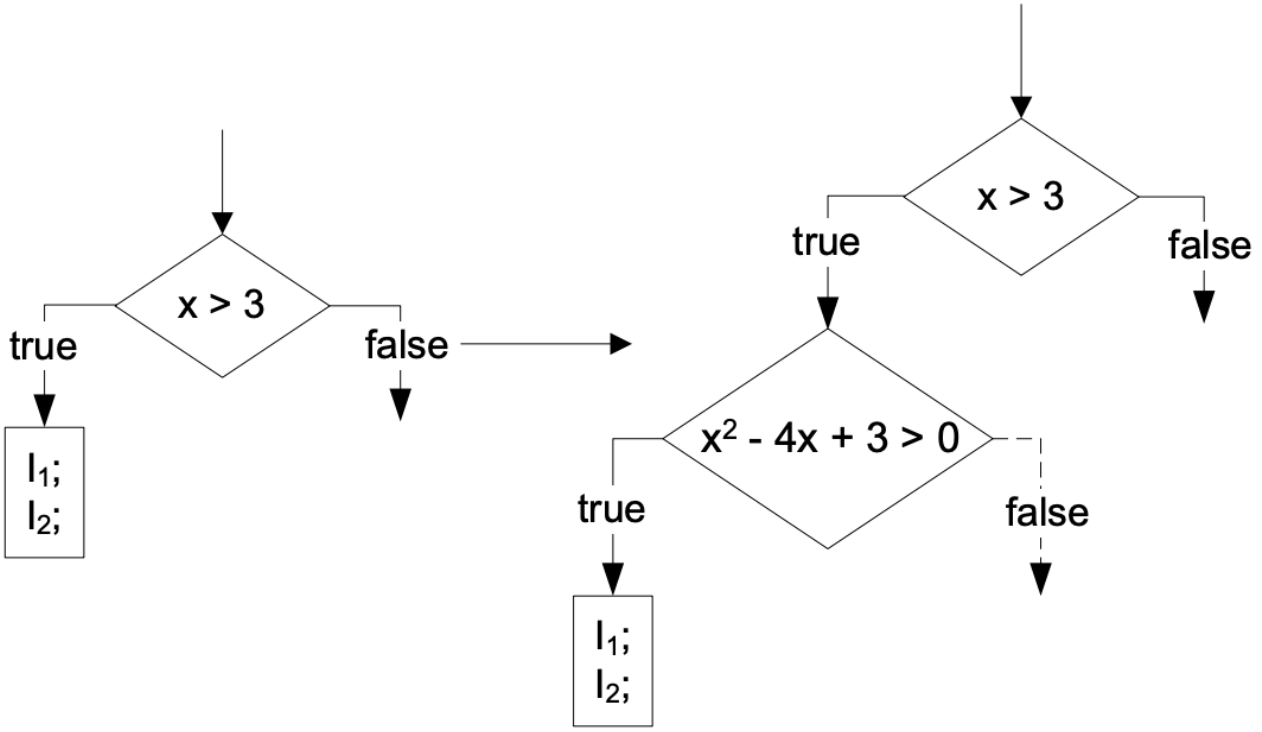
Opaque Predicates

- Invariant opaque predicates are easy



Opaque Predicates

- Context opaque predicates



Opaque Predicates based on multi-threading

- One thread puts random numbers > `n` into global data structure
- Another thread assigns `x` one of these numbers
- Then conditional

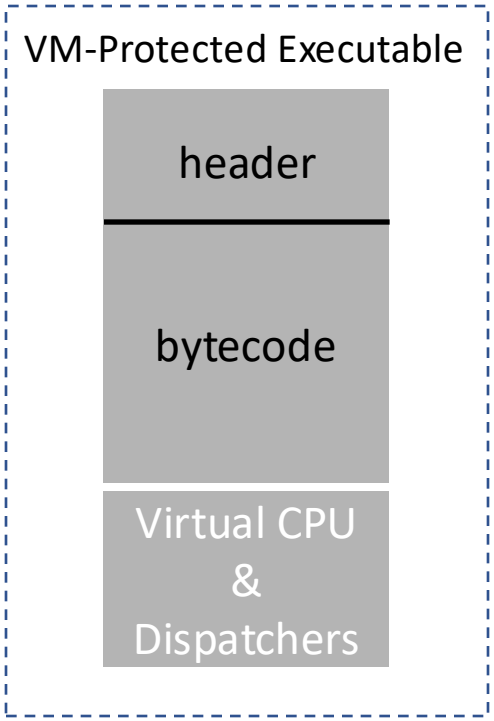
  ```
  if (x < n) …
  ```

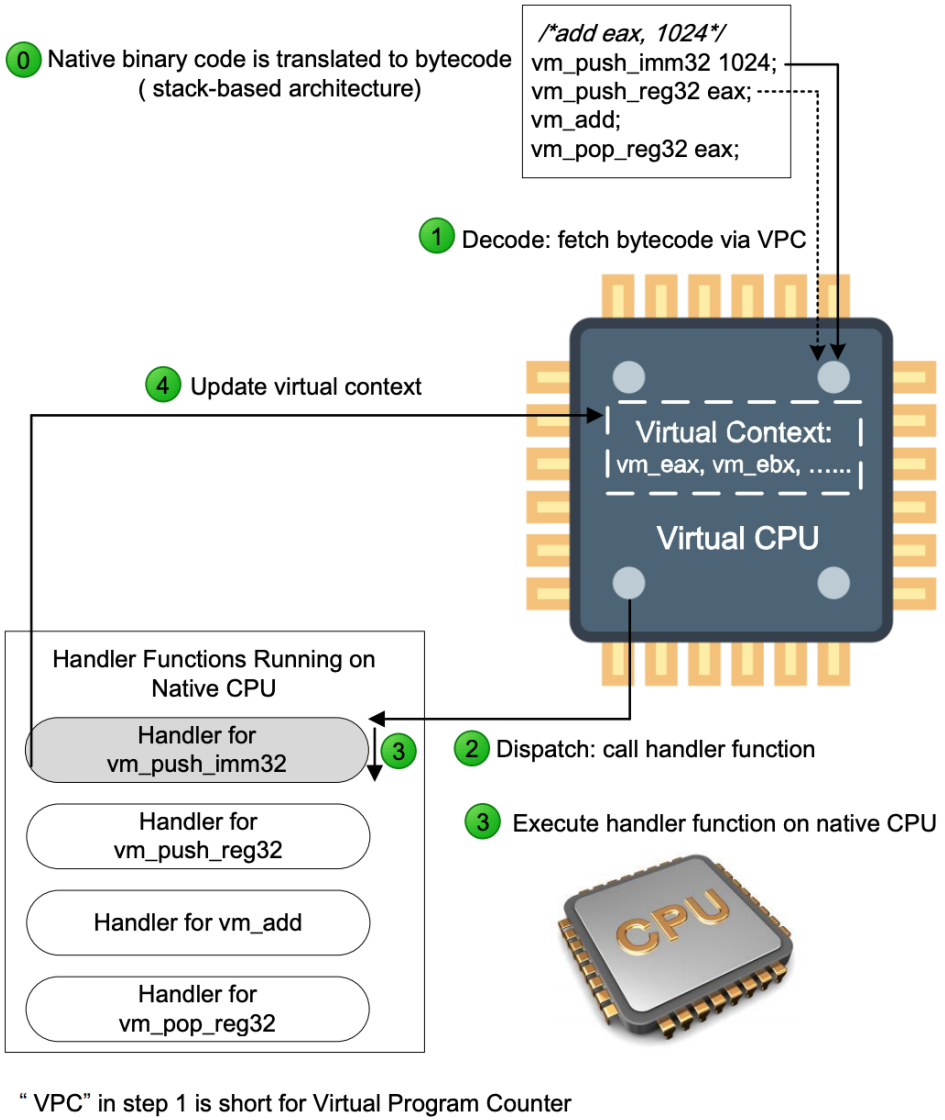
  is an opaque predicate

Advanced Method: Virtual Machine (VM)-Based Obfuscation

- Virtual machine can also be used to obfuscate software.
  - And has been commercialized very well.
  - Invent new instruction sets with secret encodings

Advanced Method: Virtual Machine (VM)-Based Obfuscation





Not a fool-proof technique,
but making analysis more difficult.

Advanced Method: Multilingual obfuscation

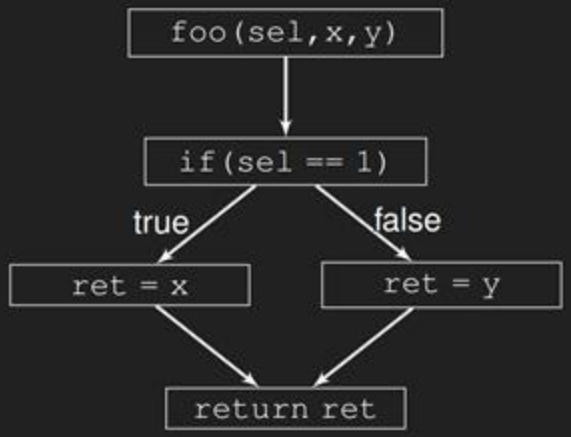
- Obfuscate programs by “mixing" highly abstract computation models
  - C/C++/Python/Java: imperative languages
  - Haskell/Scala: functional languages
  - Prolog: logic languages
  - They have different computation models!
    - Imperative language: Turing machine
    - Functional language: lambda calculus
    - Logic language: horn logic

Just for your information, details are not required.

After translation and compilation, the mixed computation model makes the *machine code* obscure

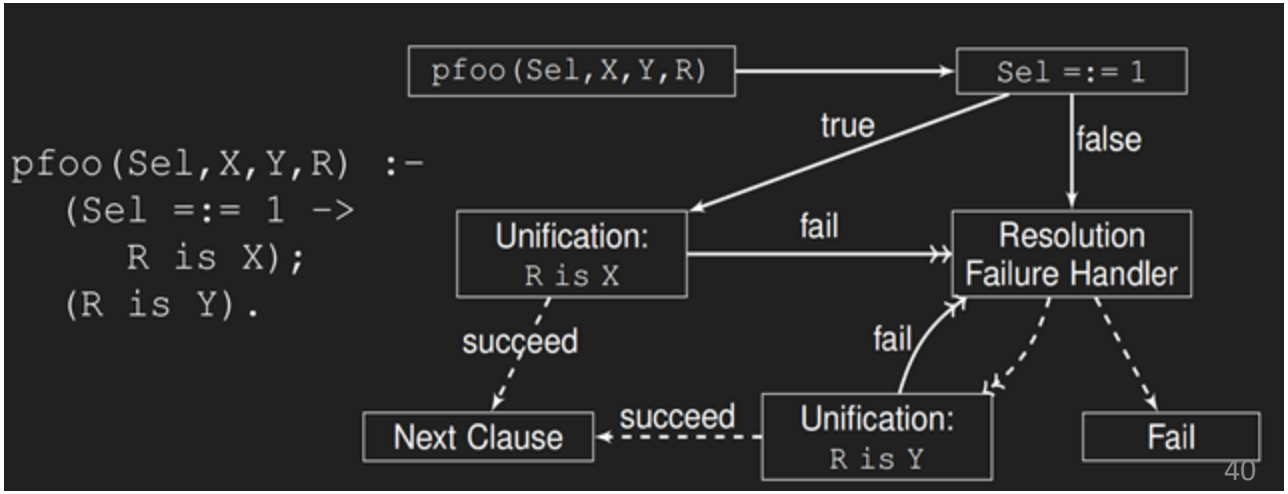
Obfuscation Effects

```
int foo(int sel,
        int x, int y)
{
  int ret;
  if(sel==1)
    ret=x;
  else
    ret=y;
  return ret;
}
```



Logic language use ***unification*** and ***backtracking*** *for computation*

```
pfoo(Sel,X,Y,R) :-
  (Sel =:= 1 ->
     R is X);
  (R is Y).
```



No need to understand the details.

Evaluation of Obfuscation Effectiveness

Four aspects

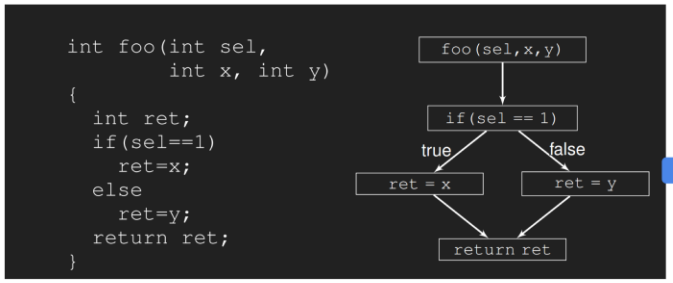
- Potency
  - How much complexity the obfuscation introduces (use software complexity Metrics to measure)
  - E.g., number of call graph edges, basic blocks, and control flow graph edges
- Stealth
  - How well obfuscated code blends in with the rest of the program
  - E.g., instruction distribution anomaly test
- Resilience
  - How difficult to undo obfuscation
  - E.g., use diffing techniques to test similarity between obfuscated and original binaries
- Cost
  - Performance penalty caused by obfuscation
  - E.g., size, execution slowdown

But good obfuscation scheme is hard to design…

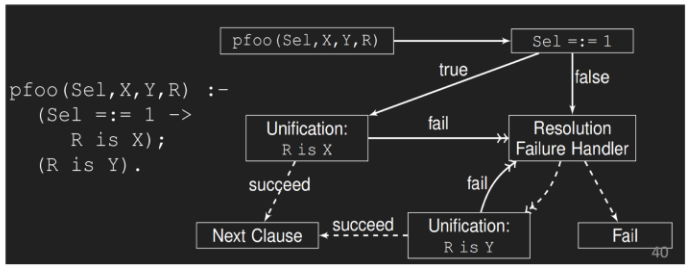
Come up with "interesting" obfuscation mutation is not too difficult, requires some creativity.

Come up with "secure" obfuscation mutation is very difficult, "security" in some sense is objective.

- Especially when people are aware of your obfuscation method!!!
- Open design with no real secret and limited search space is not good in principle.



Logic language use unification and backtracking for computation



"*Easy for me to reverse engineering*" – a reverse engineering guru commented so…