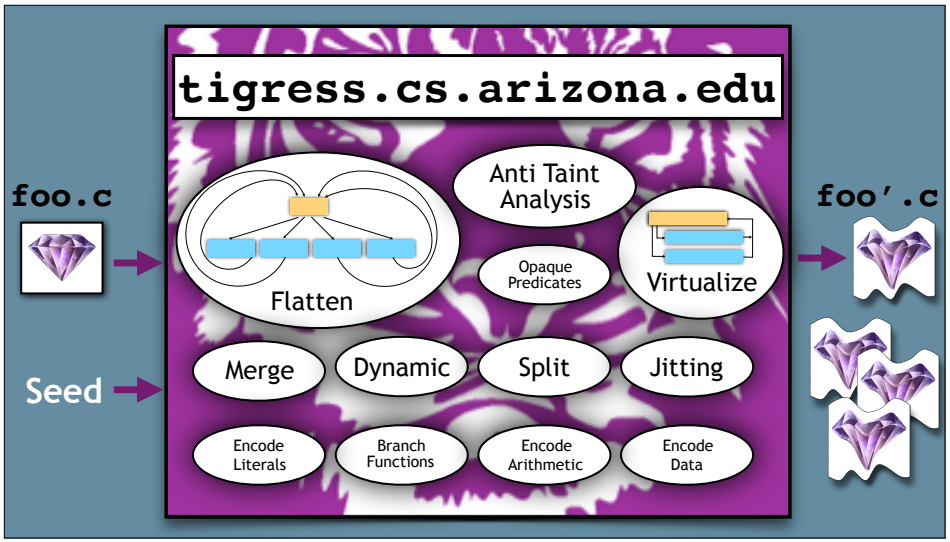
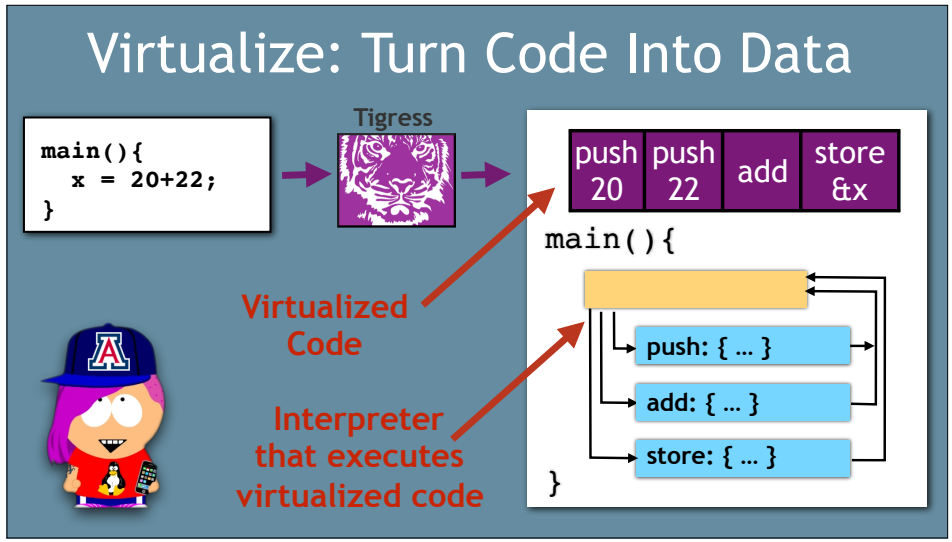
Obfuscation and Tamperproofing

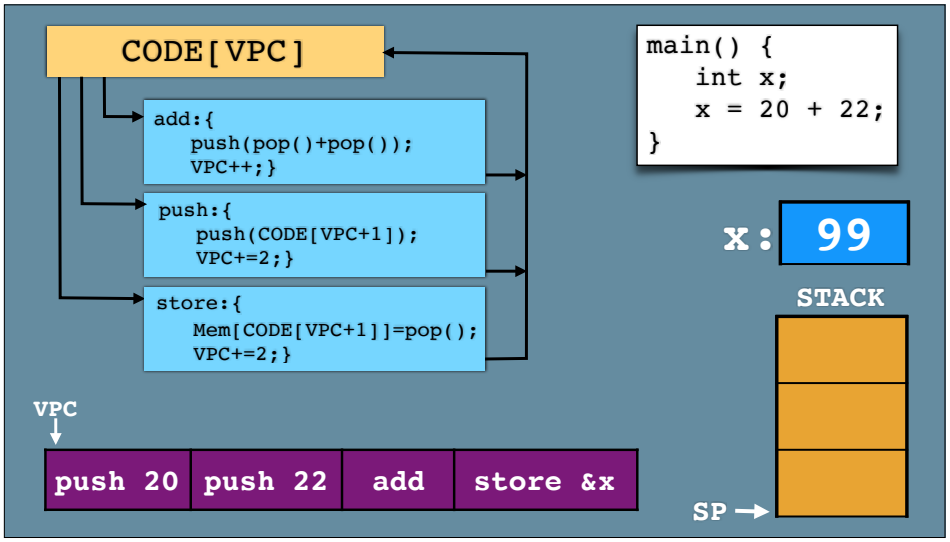
Lecture #4

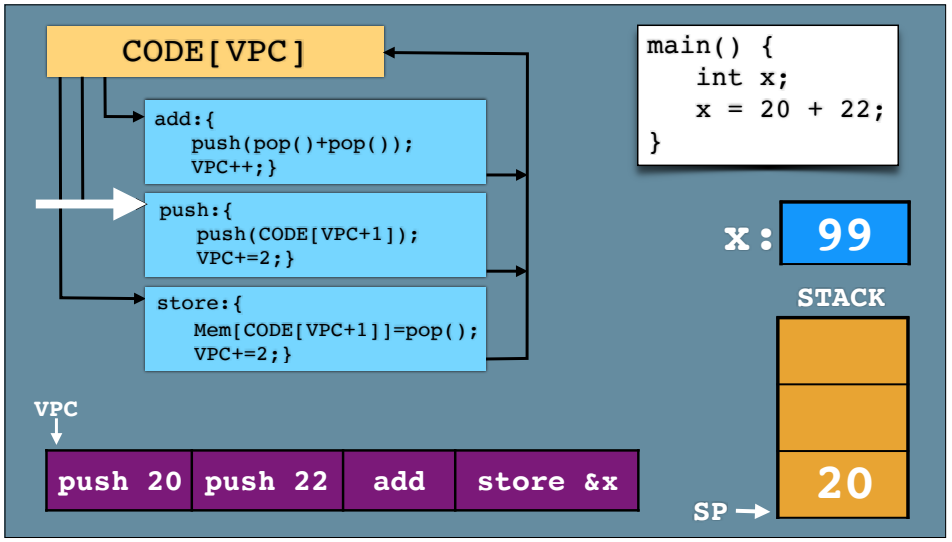
Christian Collberg
University of Arizona

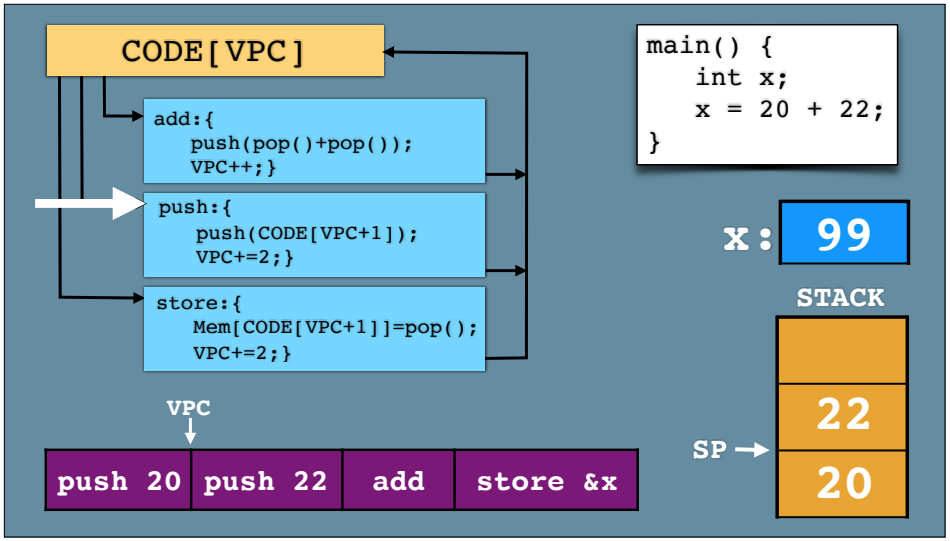
Obfuscation by Virtualization

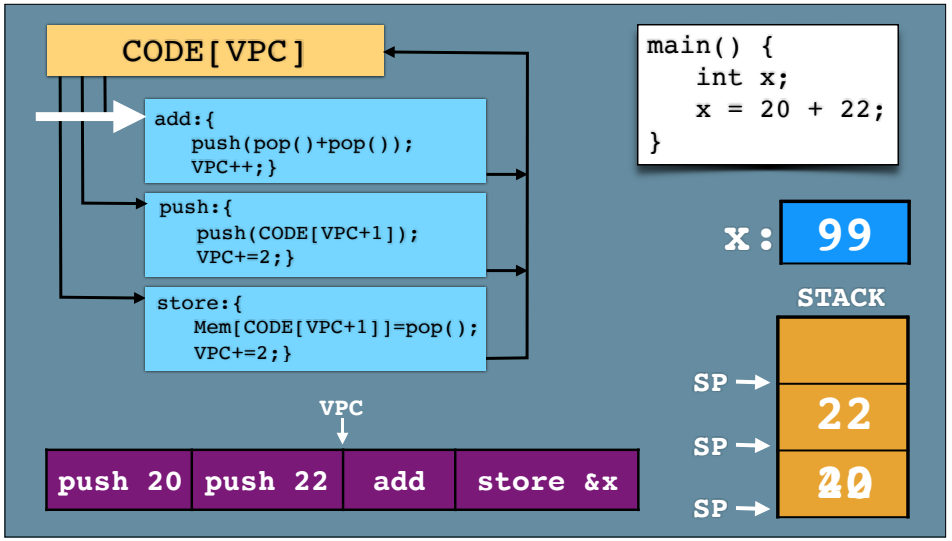


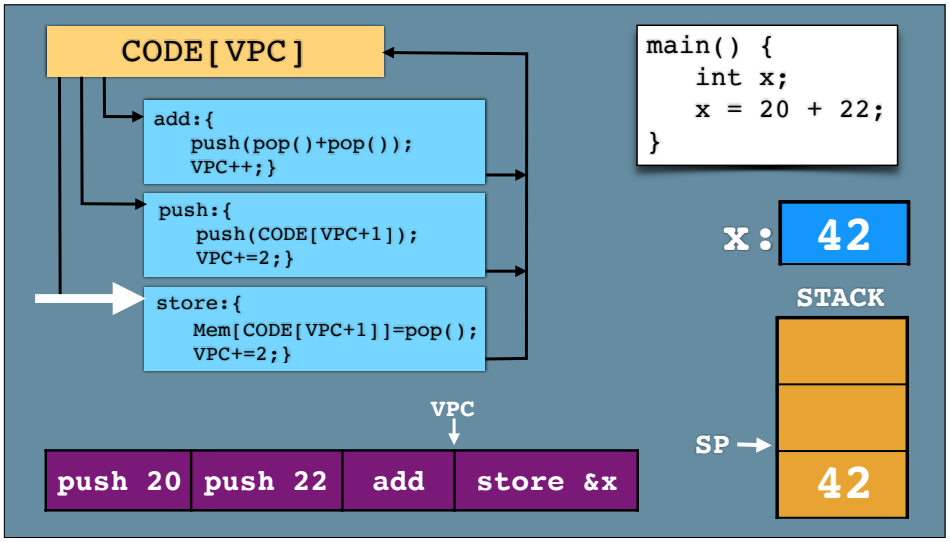



CODE[VPC]
add:{
push(pop()+pop());
VPC++;}
push:{
push(CODE[VPC+1]);
VPC+=2;}
store:{
Mem[CODE[VPC+1]]=pop();
VPC+=2;}
main() {
int x;
x = 20 + 22;
}
x: 99
STACK
VPC
push 20
push 22
add
store &x
SP


CODE[VPC]
add:{
push(pop()+pop());
VPC++;}
push:{
push(CODE[VPC+1]);
VPC+=2;}
store:{
Mem[CODE[VPC+1]]=pop();
VPC+=2;}
main() {
int x;
x = 20 + 22;
}
x: 99
STACK
20
VPC
push 20
push 22
add
store &x
SP


CODE[VPC]
add:{
push(pop()+pop());
VPC++;}
push:{
push(CODE[VPC+1]);
VPC+=2;}
store:{
Mem[CODE[VPC+1]]=pop();
VPC+=2;}
main() {
int x;
x = 20 + 22;
}
x: 99
STACK
22
20
VPC
push 20
push 22
add
store &x
SP


CODE[VPC]
add:{
push(pop()+pop());
VPC++;}
push:{
push(CODE[VPC+1]);
VPC+=2;}
store:{
Mem[CODE[VPC+1]]=pop();
VPC+=2;}
main() {
int x;
x = 20 + 22;
}
x: 99
STACK
22
20
VPC
push 20
push 22
add
store &x
SP
SP
SP



Obfuscating Arithmetic

Encoding Integer Arithmetic

$$x+y = x-\neg y-1$$
$$x+y = (x \oplus y)+2\cdot(x \wedge y)$$
$$x+y = (x \vee y)+(x \wedge y)$$
$$x+y = 2\cdot(x \vee y)-(x \oplus y)$$

Example

One possible encoding of

z=x+y+w

is

```
z = (((x ^ y) + ((x & y) << 1)) + w) +
    (((x ^ y) + ((x & y) << 1)) & w);
```

Many others are possible, which is good for diversity.

Exercise!

- The virtualizer's **add** instruction handler could still be identified by the fact that it uses a + operator!
- Try adding am arithmetic transformer:

```
tigress --Environment=x86_64:Linux:Gcc:4.6\
   --Transform=Virtualize \
      --Functions=fib \
      --VirtualizeDispatch=switch\
   --Transform=EncodeArithmetic \
      --Functions=fib \
   --out=fib5.c fib.c
```

- What differences do you notice between before and after arithmetic encoding?

```
int fib(int n ) {
  ...
  while (1) {
    switch (*(_1_fib_$pc[0])) {
    case PlusA: {
    (_1_fib_$sp[0] + -1)->_int =
       (_1_fib_$sp[0] + -1)->_int    +
       (_1_fib_$sp[0] + 0)->_int;
   break;
    }
}
```

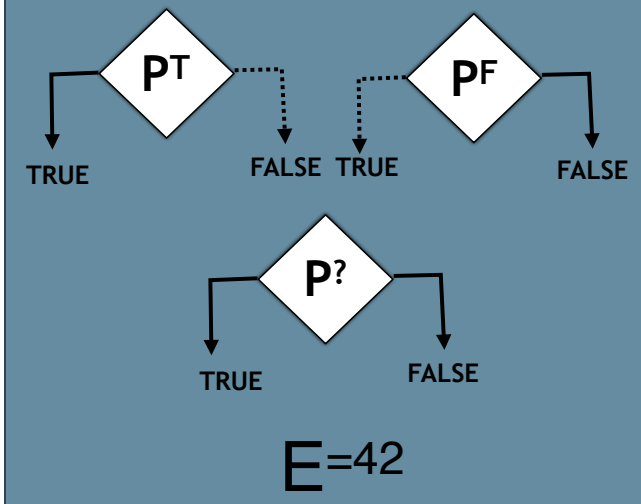
```
int fib(int n ) {
  ...
  while (1) {
    switch (*(_1_fib_$pc[0])) {
    case PlusA: {
    (_1_fib_$sp[0] + -1)->_int =
       ((_1_fib_$sp[0] + -1)->_int    ^
       (_1_fib_$sp[0] + 0)->_int)     +
       (((_1_fib_$sp[0] + -1)->_int   &
       (_1_fib_$sp[0] + 0)->_int) << 1);
    break;
    }
}
```

$$x+y = (x \oplus y)+2\cdot(x \wedge y)$$

Opaque Expressions

Opaque Expressions

An expression whose value is known to you as the defender (at obfuscation time) but which is difficult for an attacker to figure out

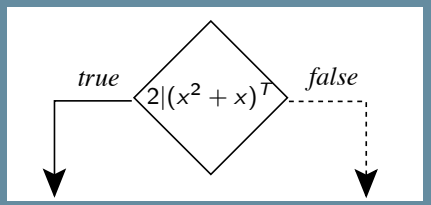


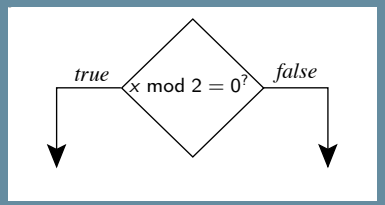
Opaquely true/ false predicate

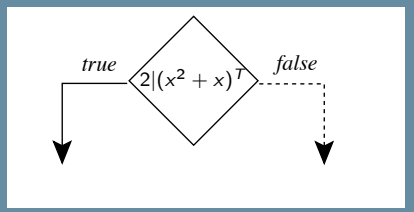
Opaquely indeterminate predicate

An expression of value v

Examples







Inserting Bogus Control Flow

```
if (x[k] == 1)
   R = (s*y) % n
else
   R = s;
s = R*R % n;
L = R;
```

```
if (x[k] == E=1)
   R = (s*y) % n
else
   R = s;
s = R*R % n;
L = R;
```

Inserting Bogus Control Flow

```
if (x[k] == 1)
   R = (s*y) % n
else
   R = s;
s = R*R % n;
L = R;
```

```
if (x[k] == 1)
   R = (s*y) % n
else
   R = s;
if (expr=T)
   s = R*R % n;
else
   s = R*R * n;
L = R;
```

Inserting Bogus Control Flow

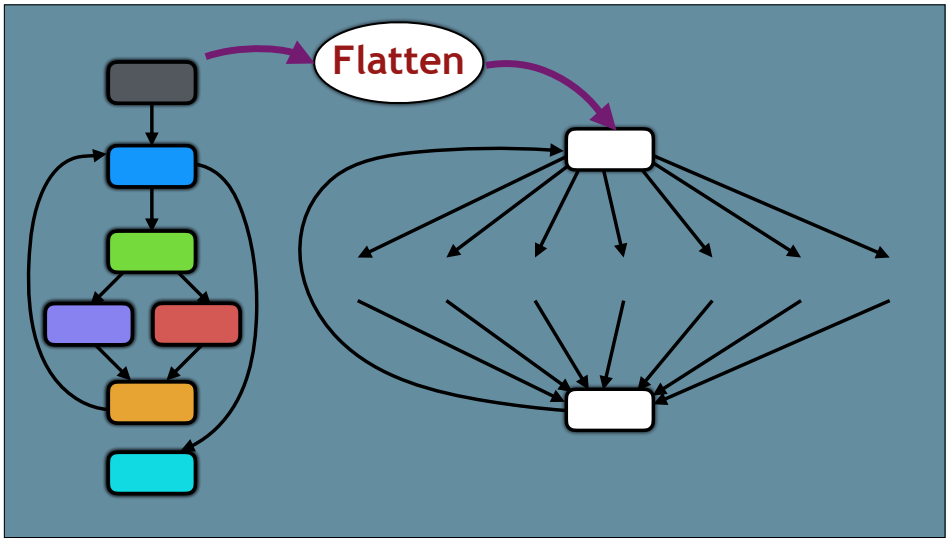
```
if (x[k] == 1)
   R = (s*y) % n
else
   R = s;
s = R*R % n;
L = R;
```

```
if (x[k] == 1)
   R = (s*y) % n
else
   R = s;
if (expr=?)
   s = R*R % n;
else
   s = (R%n)*(R%n)%n;
L = R;
```

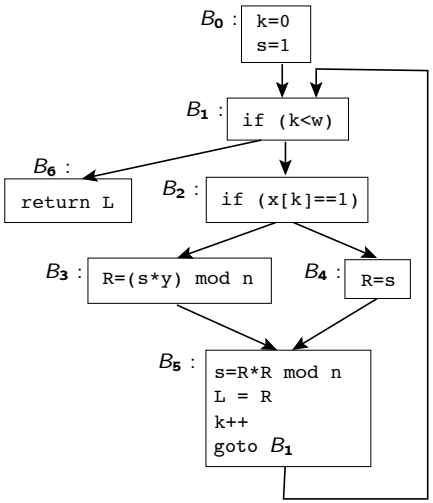
Exercise!

```
************************************************************
* 1) Opaque Predicates
************************************************************
tigress --Environment=x86_64:Linux:Gcc:4.6 --Seed=0 \
   --Transform=InitEntropy \
      --InitEntropyKinds=vars \
   --Transform=InitOpaque \
      --Functions=main\
      --InitOpaqueCount=2\
      --InitOpaqueStructs=list,array \
   --Transform=AddOpaque\
      --Functions=fib\
      --AddOpaqueKinds=question \
      --AddOpaqueCount=10 \
    --out=fib1.c fib.c
```

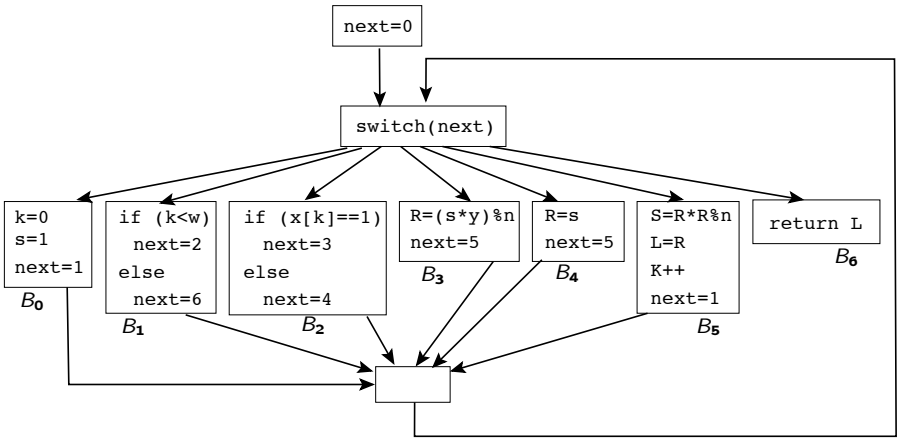
Control Flow Flattening




```
int modexp(
    int y,int x[],
    int w,int n){
    int R, L;
    int k=0; int s=0;
    while (k < w) {
        if (x[k] == 1)
            R = (s*y) % n
        else
            R = s;
        s = R*R % n;
        L = R;
        k++;
    }
    return L;
}
```

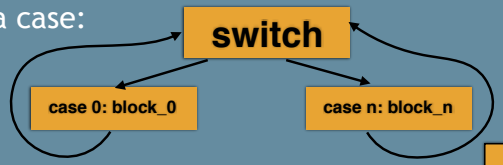



```
int modexp(int y, int x[], int w, int n) {
int R, L, k, s;
   int next=0;
   for(;;)
      switch(next) {
         case 0 : k=0; s=1; next=1; break;
         case 1 : if (k<w) next=2; else next=6; break;
         case 2 : if (x[k]==1) next=3; else next=4; break;
         case 3 : R=(s*y)%n; next=5; break;
         case 4 : R=s; next=5; break;
         case 5 : s=R*R%n; L=R; k++; next=1; break;
         case 6 : return L;
   }
}
```



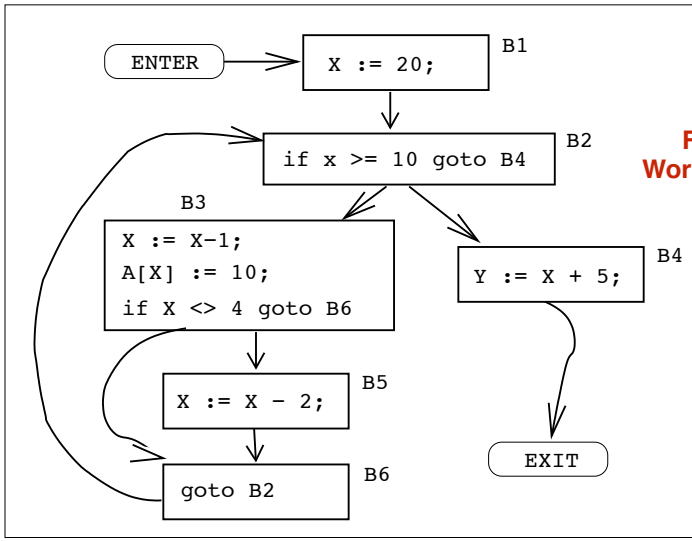
Flattening Algorithm

1. Construct the CFG
2. Add a new variable **int next=0;**
3. Create a switch inside an infinite loop, where every basic block is a case:



4. Add code to update the **next** variable:

```
case n: {
  if (expression)
    next = ...
  else
    next = ...
}
```

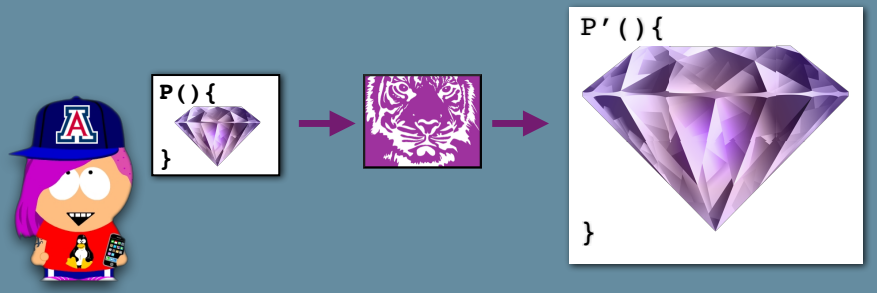


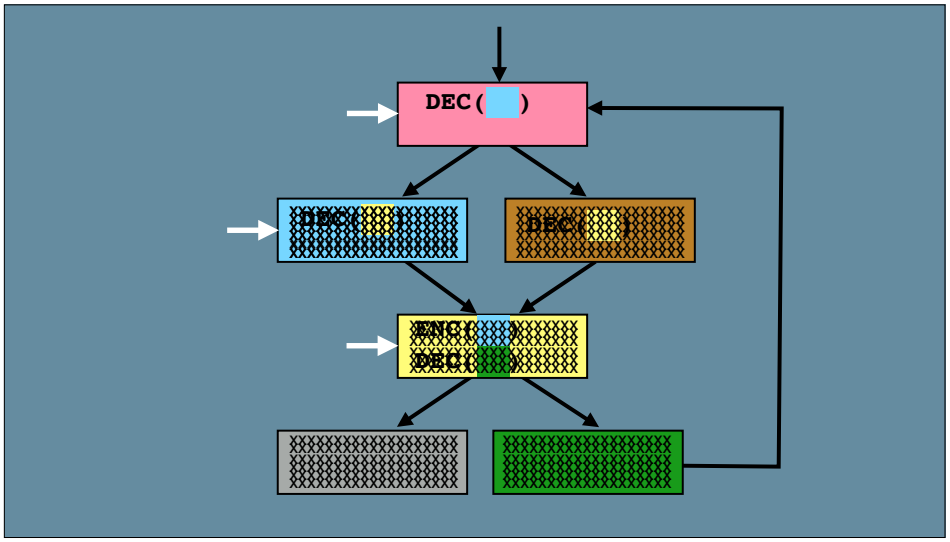
Flatten this CFG!
Work with your friends!

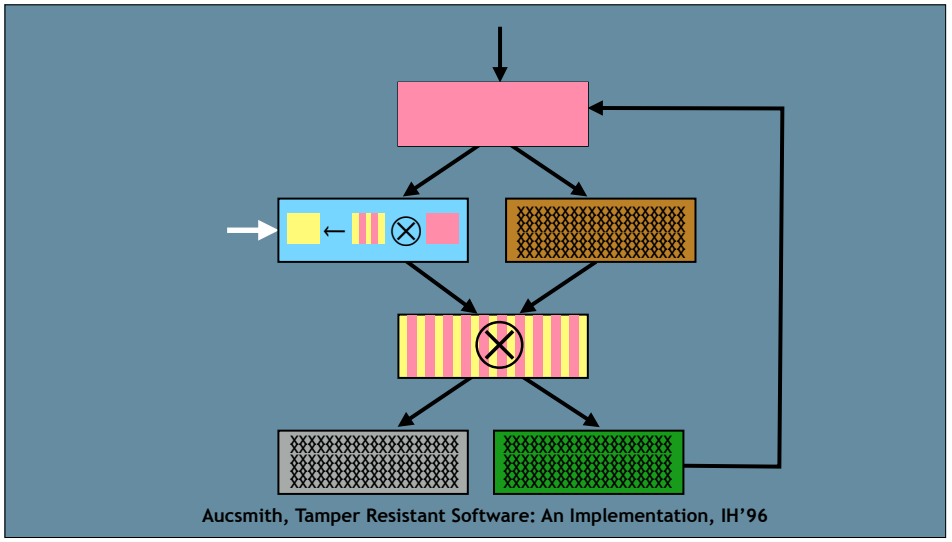
Dynamic Obfuscation

Dynamic Obfuscation

- Use self-modifying code
- Keep the code in constant flux
- The code should never exist in cleartext







Exercise!

```
*************************************************************
* 6) Dynamic Obfuscation
*************************************************************
tigress --Environment=x86_64:Linux:Gcc:4.6 \
   --Transform=JitDynamic \
      --Functions=fib \
         --JitDynamicCodecs=xtea \
         --JitDynamicDumpCFG=false \
         --JitDynamicBlockFraction=%50 \
         --out=fib6.c fib.c
```

- **Please add this to fib.c:**
  #include "tigress_unstable/jitter-amd64.c"

```
// #include "apple.h"
#include<stdio.h>
#include<stdlib.h>
#include "tigress_unstable/jitter-amd64.c"

int fib(int n) {
   int a = 1;  int b = 1; int i;
   for (i = 3; i <= n; i++) {
      int c = a + b; a = b; b = c;
   };
   return b;
}

int main(int argc, char** argv) {
  if (argc != 2) {
     printf("Give one argument!\n"); abort();
  };
  long n = strtol(argv[1],NULL,10);
  int f = fib(n);
  printf("fib(%li)=%i\n",n,f);
}
```

http://tigress.cs.arizona.edu/fib.c

Anti

Tamper

Tamperproofing has to do two things:

1. detect tampering
2. respond to tampering

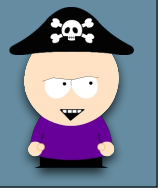
Essentially:

```
if (tampering-detected())
    respond-to-tampering()
```

but this is too unstealthy!

```
int foo () {
   if (today > "Aug 17,2016"){
     printf("License expired!");
     abort;
   }
}
```

```
check(){
 if (hash(foo)!=42)
   abort()
}
```



```
int foo() {
   … … … …
}
```

```
int foo_copy() {
   … … … …
}
```

Checker1

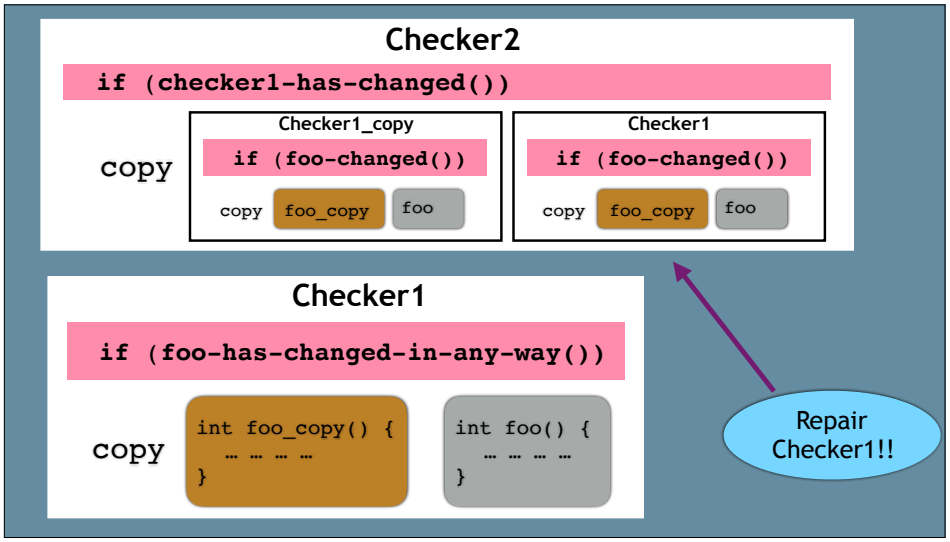
```
if (foo-has-changed-in-any-way())
```

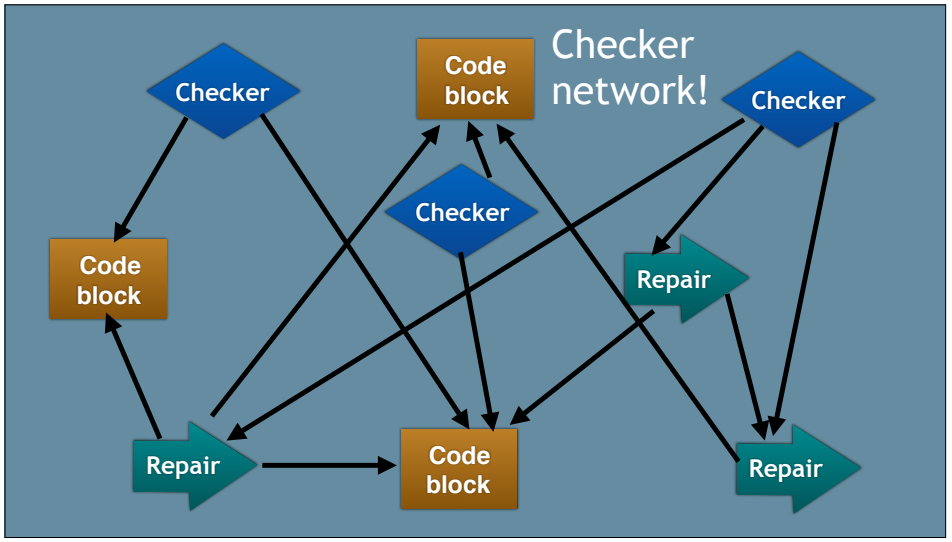
copy

```
int foo_copy() {
   … … … …
}
```

```
int foo() {
   … … … …
}
```

Repair foo!!!


Checker2
if (checker1-has-changed())
copy
Checker1_copy
if (foo-changed())
copy
foo_copy
foo
Checker1
if (foo-changed())
copy
foo_copy
foo
Checker1
if (foo-has-changed-in-any-way())
copy
int foo_copy() {
… … … …
}
int foo() {
… … … …
}
Repair
Checker1!!


Checker
Code block
Checker network!
Checker
Checker
Code block
Repair
Repair
Code block
Repair