CODE OBFUSCATION

DEFENSE STRATEGIES

Roberto Giacobazzi

Dipartimento di Informatica
Università degli Studi di Verona
Italy

ASP 2009

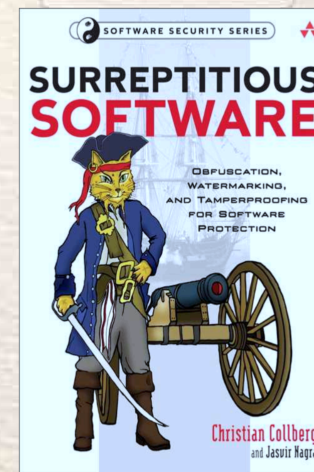
giaco CONFIDENTIAL

THE SOURCE

Most of the slides are taken from: Ch. 4, 5 and 6 of

Surreptitious Software: Obfuscation, Watermarking, and Tamperproofing for Software Protection

Christian Collberg
Jasvir Nagra
ISBN-10: 0321549252
ISBN-13: 9780321549259
Addison-Wesley Professional 2010, 792 pp.



and

Roberto Giacobazzi. **Hiding Information in Completeness Holes**.
The 6th IEEE International Conferences on
Software Engineering and Formal Methods, SEFM'08, pages 7-20, IEEE Press.

giaco CONFIDENTIAL

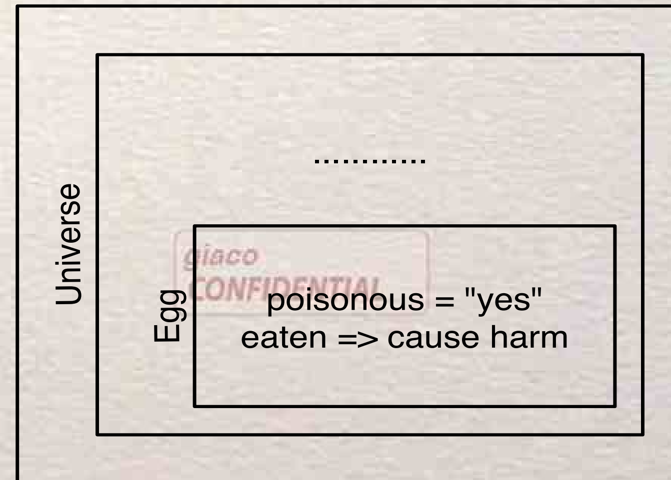
Defense Strategies

Actions in Frames: ... let us see the *Bofo marinus*

One or more slots: $name = value$

Slots may contain other slots

Conditional actions $\Rightarrow$ can take place



DEFENSE STRATEGIES: THE PRIMITIVES

We define 10 primitives which can be composed to design a generic defense strategy:

Cover

Duplicate

Split and Merge

Reorder

Map

Indirect

Mimic

Advertise

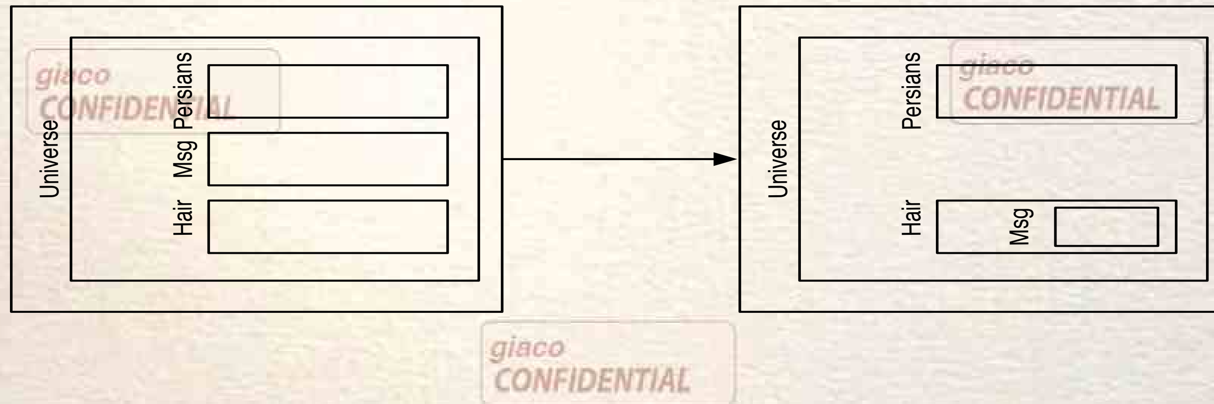
Detect/Respond

Dynamic

giaco CONFIDENTIAL

Defense Strategies: *Cover*

Given two frames x and y make x an element of y:

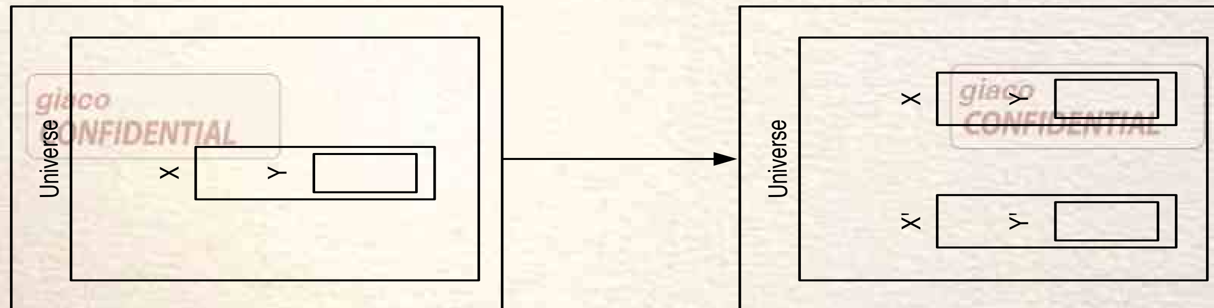


- *Cover* can be applied multiple times to hide information in an inner level
- Hiding = covering = obscuring
- Typical examples:
  - hiding keys or SW in hardened boxes (Military)
  - hiding watermarks in standard data-structures
  - hiding watermarks in images of media

giaco CONFIDENTIAL

DEFENSE STRATEGIES: *Duplicate*

Given a frame x, create a deep copy of x (keeping names unique):

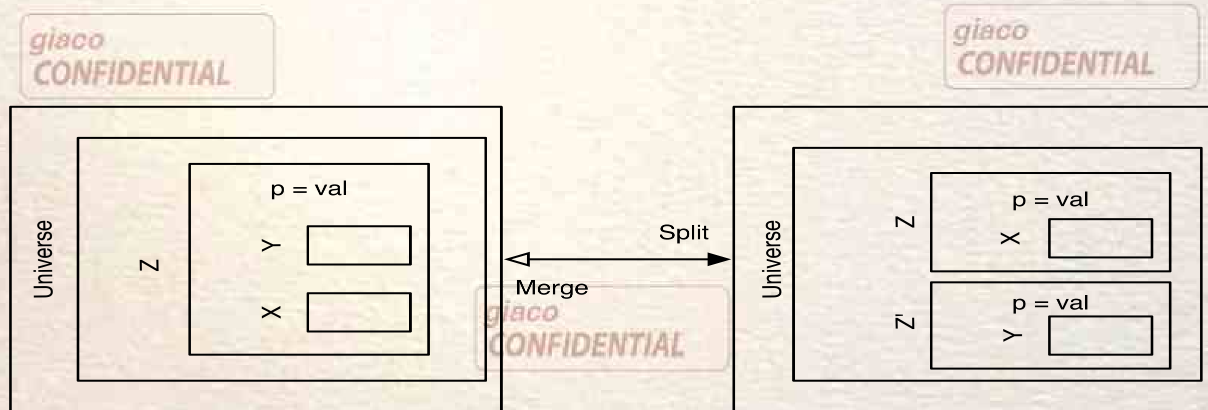


- Idea 1: Copy as *decoy*: make the universe larger and harder to scan
- Idea 2: Copy as *reduplication*: make the universe larger, full of your copies (signatures)
- Typical examples:
  - reduplication for protecting its own DNA
  - dummy targets for confusing adversaries
    `int THE_WATERMARK_IS_HERE = 666`
  - obfuscate f by $f' := Duplicate(f)$, $f'' := Obf(f')$ and call f and f''

giaco CONFIDENTIAL

DEFENSE STRATEGIES: *Split & Merge*

Given a predicate π and a frame z create a new frame z' such that z' has all the properties of z and $\forall x \in z'.\ \pi(x)$. Merge is set union of frames (and related properties):



Typically used in combination: take two functions f and g: $\mathit{Split}(f) = (f_1, f_2)$ and $\mathit{Split}(g) = (g_1, g_2)$, then:

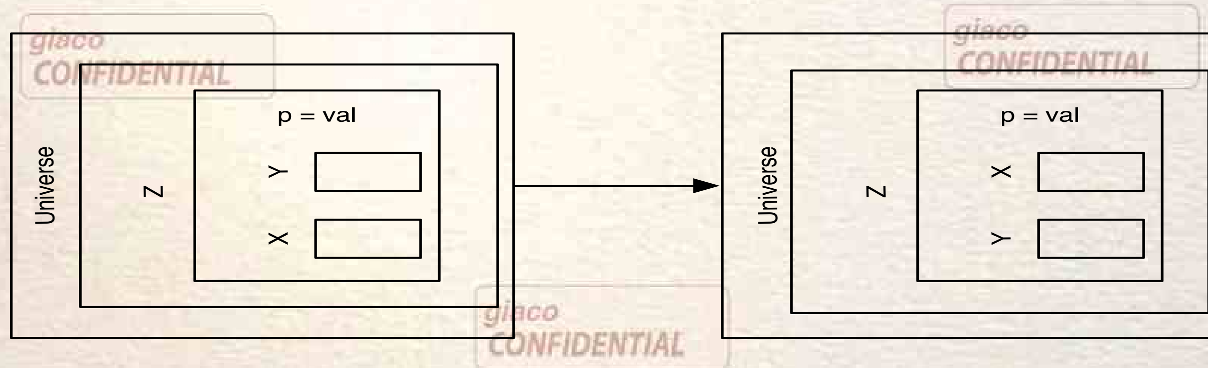
$$fg = \mathit{Merge}(f_1, g_2) =$$
$$\lambda x, y.\ \texttt{if}\ x = 1\ \texttt{then}\ f_1(y)\ \texttt{else}\ g_2(y)$$
$$gf = \mathit{Merge}(f_2, g_1) =$$
$$\lambda x, y.\ \texttt{if}\ x = 2\ \texttt{then}\ f_2(y)\ \texttt{else}\ g_1(y)$$

$$\Rightarrow\ \texttt{call}\ f(y) \longrightarrow \left[\begin{array}{l} x = 1 \\ fg(x, y) \\ x{+}{+} \\ gf(x, y) \end{array}\right.$$

giaco CONFIDENTIAL

DEFENSE STRATEGIES: *Reorder*

Given a frame z and a permutation function f, reorder the elments of z according to f:

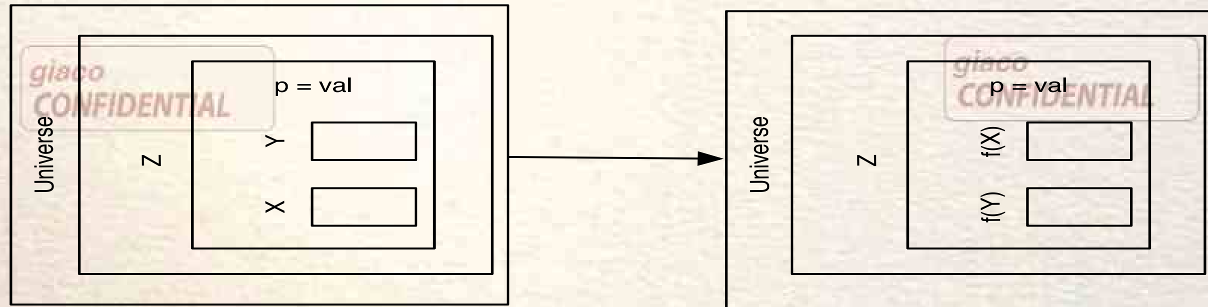


- Used in early SW watermarking e.g., by reordering basic blocks in CFG
- Used in code obfuscation and metamorphism by reordering basic blocks in CFG

giaco CONFIDENTIAL

Defense Strategies: *Map*

Given a frame x and a function f, replace every element e in x with $f(e)$:



Implements Security-through-obscurity: translation, crypto, etc.

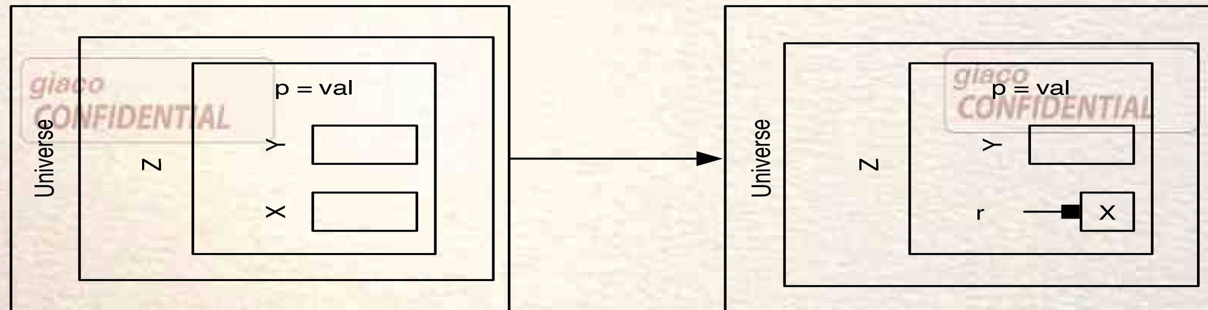
Protect confidentiality by name obfuscation (translation): variables, functions, data

giaco CONFIDENTIAL

giaco CONFIDENTIAL

DEFENSE STRATEGIES: *Indirect*

Given a frame x add an indirect reference r to x:



The cost of following pointers makes the analysis harder!!

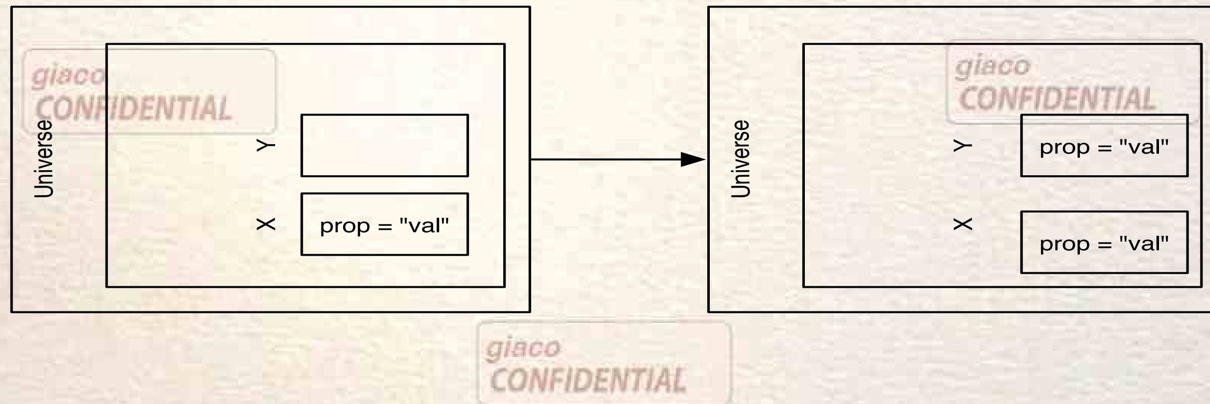
```
void fun1(){}
void fun2(){
...
    fun1();
}
```

```
void fun1(){}
void (*ref1)()=&fun1;
void (*ref2)()=&ref1;
void fun2(){
 (**ref2)();
}
```

Combined with *map* you can hide references.

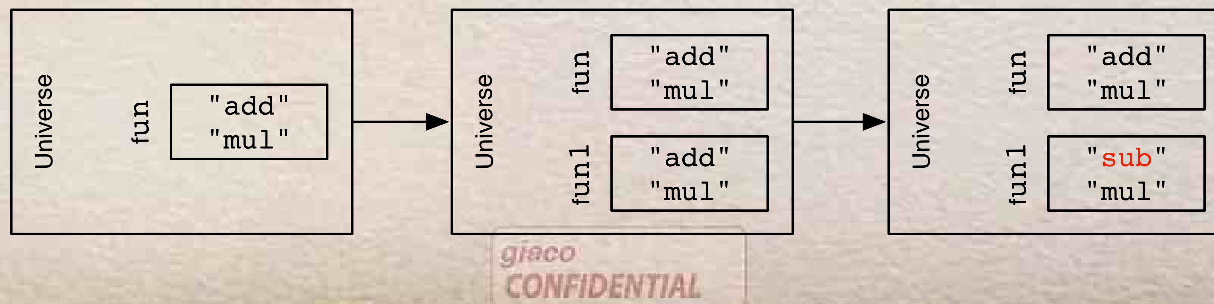
DEFENSE STRATEGIES: *Mimic*

Given two frames x and y, where x holds a property `prop`, copy `prop` into y:



Common in wild-life (animals)

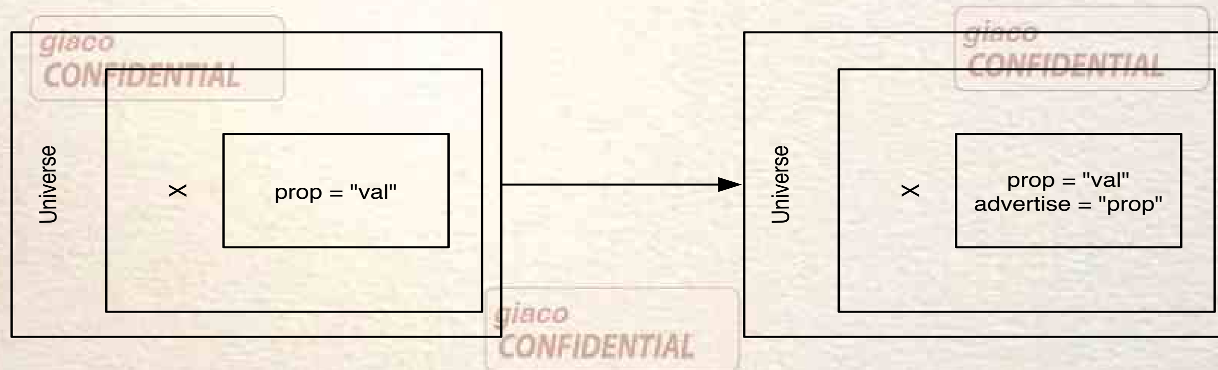
Fundamental for stealthy: the new (watermarked) code must resemble the same a standard code. Here is a static watermarking:



giaco CONFIDENTIAL

Defense Strategies: *Advertise*

Given a frame x with a property `prop` add a property `advertise` to x with value `prop`:

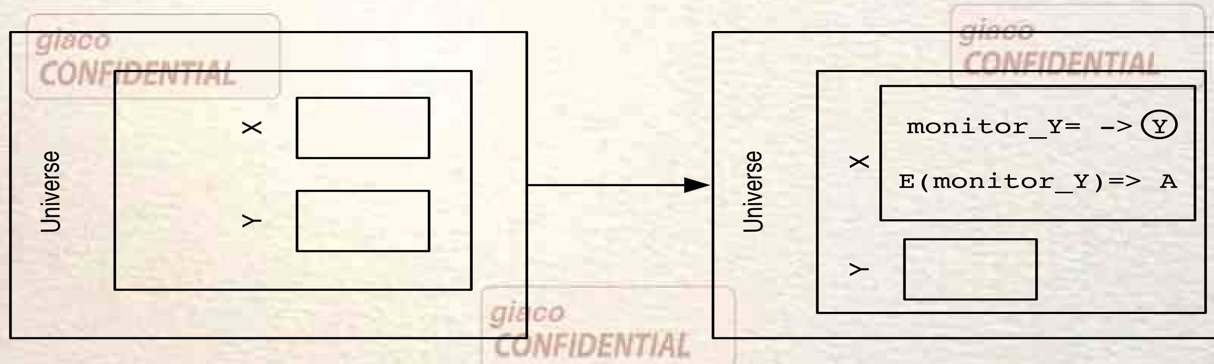


- Opposite to security-through-obscurity
- Openly display a situation in order to discourage attacks
- Example by false advertising: Say that P is watermarked when it is not!

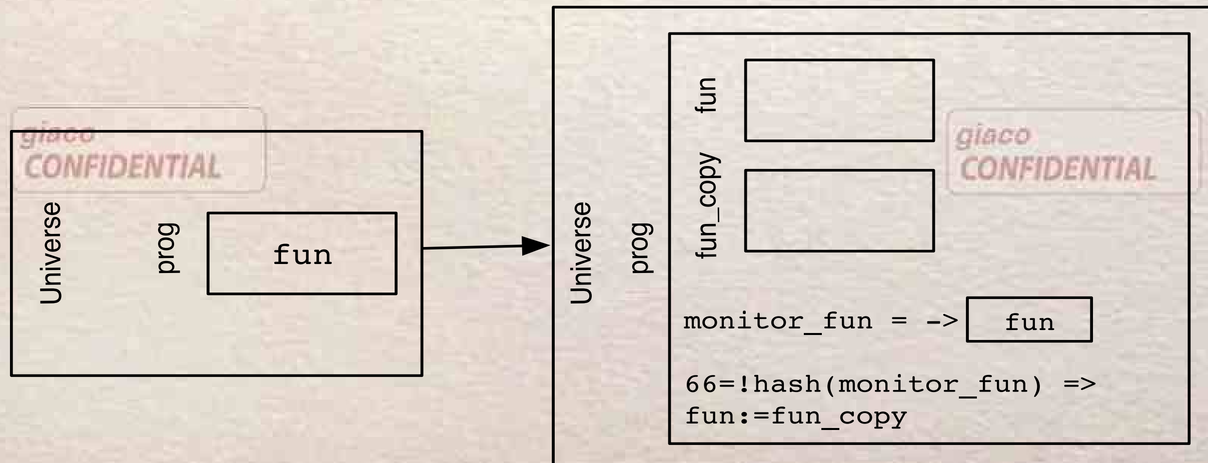
giaco CONFIDENTIAL

DEFENSE STRATEGIES: *Detect/Respond*

Given two frames x and y add a demon to x that exectutes action `A` if event `E` happens to y:



Typical in tamper-proofing: *Duplictate* $\rightarrow$ *Map* $\rightarrow$ *Detect/Respond*



giaco CONFIDENTIAL

Defense Strategies: *Dynamic*

Iterate a primitive f over a frame x generating a sequence (finite or infinite) of frames:

$$f(x) \longrightarrow f(f(x)) \longrightarrow f(f(f(x))) \longrightarrow ....$$

where

$f \in \{Cover, Duplicate, Split, Merge, Reoder, Map, Indirect, Mimic, Advertise, Detect/Respond\}$

Dynamic change for confusing the attacker: *Polymorphism* and *Metamorphism*

Useful in polymorphic malware: dynamically decrypt (map) chunks of code, execute, rencrypt (map)

giaco CONFIDENTIAL

THE PROBLEM: OBFUSCATION VS DIVERSITY

F. Cohen, Operating systems protection through program evolution, 1993

Generate syntactically different semantic equivalent programs

P and Q are syntactically different if $P \neq Q$

P and Q are semantic equivalent if $P \equiv Q$ i.e.,

- ✔ If $[\![P]\!](x)\downarrow$ and $[\![Q]\!](x)\downarrow$ then $[\![P]\!](x) = [\![Q]\!](x)$
- ✔ If $[\![P]\!](x)\uparrow$ or $[\![Q]\!](x)\uparrow$ then $[\![P]\!](x)\uparrow$ and $[\![Q]\!](x)\uparrow$

Given a program P:

$$\mathfrak{D}(P) \stackrel{\text{def}}{=} \left\{ \, Q \, \middle| \; P \equiv Q \wedge P \neq Q \; \right\}$$ is a non recursive set!

giaco CONFIDENTIAL

Layout obfuscation

Standard and easy methods for making your code diverse:

Change your code by substituting equivalent expressions:

```
y = x * 42   ⟹   y = x << 5;
                 y+= x <<3;
                 y+= x<<1;
```

Reordering code: break locality which is a typical principle used in reverse engineering!

Identifier renaming: ... that is really too easy!!!

giaco CONFIDENTIAL

Obfuscation by interpretation

see Y. Futamura, Partial Evaluation of Computation Process, 1971

Consider two programming languages (abstract machines): $\mathcal{S}$ and $\mathcal{T}$ with common data

Interpreter: $\forall \ell \in \{\mathcal{S}, \mathcal{T}\}$

- ✔ $\mathtt{int}^{\mathcal{S}} : \mathcal{S}.prog \times data \longrightarrow data \cup \{\uparrow\}\,,\ \mathtt{int}^{\mathcal{S}} \in \mathcal{S}.prog$
- ✔ $\mathtt{int}^{\mathcal{T}} : \mathcal{T}.prog \times data \longrightarrow data \cup \{\uparrow\}\,,\ \mathtt{int}^{\mathcal{T}} \in \mathcal{S}.prog$
- ✔ $\forall P \in \ell.prog,\ \forall d \in data :\ [\![P]\!]^{\ell}(d) = [\![\mathtt{int}^{\ell}]\!]^{\mathcal{S}}(p, d)$

Code specializer: $\forall \ell \in \{\mathcal{S}, \mathcal{T}\}$

- ✔ $\mathtt{spec}^{\mathcal{S}} : \mathcal{S}.prog \times data \longrightarrow \mathcal{S}.prog\,,\ \mathtt{spec}^{\mathcal{S}} \in \mathcal{S}.prog$
- ✔ $\mathtt{spec}^{\mathcal{T}} : \mathcal{S}.prog \times data \longrightarrow \mathcal{T}.prog\,,\ \mathtt{spec}^{\mathcal{T}} \in \mathcal{S}.prog$
- ✔ $\forall P \in \mathcal{S}.prog,\ \forall d, s \in data :\ [\![P]\!]^{\mathcal{S}}(s, d) = [\![[\![\mathtt{spec}^{\ell}]\!]^{\mathcal{S}}(P, s)]\!]^{\ell}(d)$

Idea:

- $\mathcal{S}$ is the source language (open)
- $\mathcal{T}$ is the secrete architecture (hidden)

giaco CONFIDENTIAL

Obfuscation by interpretation

see Y. Futamura, Partial Evaluation of Computation Process, 1971

Obfuscated code for P is a compiled code from $\mathcal{S}$ to $\mathcal{T}$ and back to $\mathcal{S}$:

$$
\begin{aligned}
[\![P]\!]^{\mathcal{S}}(d) &= [\![\mathtt{int}^{\mathcal{S}}]\!]^{\mathcal{S}}(P, d) \\
&= [\![[\![\mathtt{spec}^{\mathcal{T}}]\!]^{\mathcal{S}}(\mathtt{int}^{\mathcal{S}}, P)]\!]^{\mathcal{T}}(d) \\
&= [\![\mathtt{int}^{\mathcal{T}}]\!]^{\mathcal{S}}([\![\mathtt{spec}^{\mathcal{T}}]\!]^{\mathcal{S}}(\mathtt{int}^{\mathcal{S}}, P), d) \\
&= [\![[\![\mathtt{spec}^{\mathcal{S}}]\!]^{\mathcal{S}}(\mathtt{int}^{\mathcal{T}}, [\![\mathtt{spec}^{\mathcal{T}}]\!]^{\mathcal{S}}(\mathtt{int}^{\mathcal{S}}, P))]\!]^{\mathcal{S}}(d)
\end{aligned}
$$

$$\mathtt{obf}(P) = [\![\mathtt{spec}^{\mathcal{S}}]\!]^{\mathcal{S}}(\mathtt{int}^{\mathcal{T}}, [\![\mathtt{spec}^{\mathcal{T}}]\!]^{\mathcal{S}}(\mathtt{int}^{\mathcal{S}}, P)) \in \mathcal{S}.prog$$

In order to attack $\mathtt{obf}(P)$ you need to understand the relation between $\mathcal{S}$ and $\mathcal{T}$

$\mathtt{obf}(P)$ may run 10-100 time slower!!

giaco CONFIDENTIAL

giaco CONFIDENTIAL

OBFUSCATOR BY INTERPRETATION

see Y. Futamura, Partial Evaluation of Computation Process, 1971

An obfuscator can be generated by further specializing the specializer by an interpreter:

$$\begin{aligned}
\mathtt{obf}(P) &= [\![\mathtt{spec}^{\mathcal{S}}]\!]^{\mathcal{S}}(\mathtt{int}^{\mathcal{T}}, [\![\mathtt{spec}^{\mathcal{T}}]\!]^{\mathcal{S}}(\mathtt{int}^{\mathcal{S}}, P)) \\
&= [\![[\![\mathtt{spec}^{\mathcal{S}}]\!]^{\mathcal{S}}(\mathtt{spec}^{\mathcal{S}}, \mathtt{int}^{\mathcal{T}})]\!]^{\mathcal{S}}([\![\mathtt{spec}^{\mathcal{T}}]\!]^{\mathcal{S}}(\mathtt{int}^{\mathcal{S}}, P)) \\
&= [\![[\![\mathtt{spec}^{\mathcal{S}}]\!]^{\mathcal{S}}(\mathtt{spec}^{\mathcal{S}}, \mathtt{int}^{\mathcal{T}})]\!]^{\mathcal{S}}([\![[\![\mathtt{spec}^{\mathcal{S}}]\!]^{\mathcal{S}}(\mathtt{spec}^{\mathcal{T}}, \mathtt{int}^{\mathcal{S}})]\!]^{\mathcal{S}}(P))
\end{aligned}$$

$$\mathtt{obf} = [\![\mathtt{spec}^{\mathcal{S}}]\!]^{\mathcal{S}}(\mathtt{spec}^{\mathcal{T}}, \mathtt{int}^{\mathcal{S}})\,;\,[\![\mathtt{spec}^{\mathcal{S}}]\!]^{\mathcal{S}}(\mathtt{spec}^{\mathcal{S}}, \mathtt{int}^{\mathcal{T}}) \in \mathcal{S}.\mathit{prog}$$

By modifying $\mathcal{T}$ you can generate different obfuscators by interpretation!

Highly modular bt extremely expensive (x100 slowdown by emulation)

OBFUSCATOR GENERATOR BY INTERPRETATION

see Y. Futamura, Partial Evaluation of Computation Process, 1971

A generator of obfuscators can be obtained by further specializing the specializer:

$$\begin{aligned}\mathtt{obf}(P) &= [\![\mathtt{spec}^{\mathcal{S}}]\!]^{\mathcal{S}}(\mathtt{int}^{\mathcal{T}}, [\![\mathtt{spec}^{\mathcal{T}}]\!]^{\mathcal{S}}(\mathtt{int}^{\mathcal{S}}, P)) \\ &= [\![[\![\mathtt{spec}^{\mathcal{S}}]\!]^{\mathcal{S}}(\mathtt{spec}^{\mathcal{S}}, \mathtt{int}^{\mathcal{T}})]\!]^{\mathcal{S}}([\![\mathtt{spec}^{\mathcal{T}}]\!]^{\mathcal{S}}(\mathtt{int}^{\mathcal{S}}, P)) \\ &= [\![[\![\mathtt{spec}^{\mathcal{S}}]\!]^{\mathcal{S}}(\mathtt{spec}^{\mathcal{S}}, \mathtt{int}^{\mathcal{T}})]\!]^{\mathcal{S}}([\![[\![\mathtt{spec}^{\mathcal{S}}]\!]^{\mathcal{S}}(\mathtt{spec}^{\mathcal{T}}, \mathtt{int}^{\mathcal{S}})]\!]^{\mathcal{S}}(P))\end{aligned}$$

$$\mathtt{obf} = [\![\mathtt{spec}^{\mathcal{S}}]\!]^{\mathcal{S}}(\mathtt{spec}^{\mathcal{T}}, \mathtt{int}^{\mathcal{S}})\ ;\ [\![\mathtt{spec}^{\mathcal{S}}]\!]^{\mathcal{S}}(\mathtt{spec}^{\mathcal{S}}, \mathtt{int}^{\mathcal{T}}) \in \mathcal{S}.\mathit{prog}$$

By modifying $\mathcal{T}$ you can generate different obfuscators by interpretation

giaco CONFIDENTIAL

Combining obfuscators: diversification

Assume you have a set of obfuscation strategies: $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$

Let $\texttt{prob}[i]$ be the probability of applying $\mathcal{T}_i$

Let $\mathit{thresholdspace}$ be the bound to the space of obfuscated code

```
input P;
while    space(P) ≤ thresholdspace {
          x ← y ← 0
          a ← random in [0,1]
          while   x ≤ a {
                    y ← y + 1;
                    x ← x + prob[y]
                  };
          apply(T_y, P)
        }
```

giaco CONFIDENTIAL

COMBINING OBFUSCATORS: DIVERSIFICATION II

Assume you have a set of obfuscation strategies: $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$

P consists of objects $\langle s_1, \ldots \rangle$ (routines, modules, etc

Let $\texttt{prio}[s_i]$ is the importance of protecting object s_i

`acceptCost` is the maximum execution penalty allowed

`reqObf` is the amount of obfuscation required

```
input P = ⟨s1, ...⟩;
while   !done(acceptCost, reqObf) {
          s ← max in prio;
          t ← selectTrans(s, T);
          apply(t, s);
          update(prio[s])
        };
```

giaco CONFIDENTIAL

COMBINING OBFUSCATORS: DEPENDENCIES

Assume you have a set of obfuscation strategies: $\mathcal{T} = \{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$

The order by which obfuscations are applied may be relevant!

Idea: build a FSA whose language *describe* the acceptable order of obfuscations:

$\mathcal{T} = \{\mathcal{T}_1, \mathcal{T}_2\}$

where

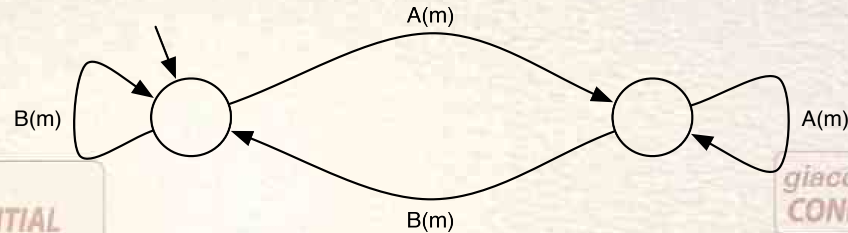
after $\mathcal{T}_1$ do $\mathcal{T}_2$

$P = \mathtt{m}()$ and $\begin{cases} Obf.level & 0.6 \\ Perf.critical & 0.2 \end{cases}$

$$\begin{bmatrix} Transf & Potency & Degradation \\ \mathcal{T}_1 & 1.0 & 0.9 \\ \mathcal{T}_2 & 0.5 & 0.3 \end{bmatrix}$$

giaco CONFIDENTIAL

COMBINING OBFUSCATORS: DEPENDENCIES



The accepted language of the FSA $\mathfrak{G}$ is: $(B(m) + A(m)^+ B(m))^*$

The best sequences can be chosen by using weights on edge:

$$\forall t \in \mathcal{T}, \forall s \in P :$$

$$weigh(t, s) = \mathcal{F}(Obf.level(s), Perf.critical(s), Potency(t), Degradation(t))$$

where $\mathcal{F} : [0, 1]^4 \longrightarrow [0.1]$ combines the single weights!

Goal: maximize the weight along some path!

giaco CONFIDENTIAL

DEFINITIONS

Let us have some secret σ in P that we want to hide: $\mathfrak{O}(P)$. What is σ?

- The source code of P;
- The module organization of P;
- Function names
- Location of critical functions (e.g., license check)
- The value of particular data (crypto keys, license etc)

Let $\mathfrak{O} : \mathbb{P} \longrightarrow \mathbb{P}$ be a program transformation. $\mathfrak{O}(P)$ is an obfuscation of P if:

- $P \equiv_\alpha \mathfrak{O}(P)$;
- σ is *hidden* in $\mathfrak{O}(P)$.

To understand how *hidden* is σ we need to know how to *attack* σ

giaco CONFIDENTIAL

giaco CONFIDENTIAL

DEFINITIONS

Let $\mathfrak{O} : \mathbb{P} \longrightarrow \mathbb{P}$ be an obfuscation. $\mathfrak{O}(P)$ is the obfuscation of $P \in \mathbb{P}$

Let $\mathcal{A}$ be an analysis for the language $\mathbb{P}$:

- If $\sigma = \mathcal{A}(P)$ then $\mathcal{A}$ reveals σ;
- $P \equiv_\alpha \mathfrak{O}(P)$;

What is the relation between $\mathcal{A}$ and $\mathfrak{O}$?

- $\mathfrak{O}$ is *effective* if $\mathcal{A}(\mathfrak{O}(P)) \neq \sigma$ or $\mathcal{A}(\mathfrak{O}(P))$ is harder than $\mathcal{A}(P)$;
- $\mathfrak{O}$ is *ineffective* if $\mathcal{A}(\mathfrak{O}(P)) = \sigma$;
- $\mathfrak{O}$ is *defective* if $\mathcal{A}(\mathfrak{O}(P)) \neq \sigma$ and $\mathcal{A}(\mathfrak{O}(P))$ is easier than $\mathcal{A}(P)$.

A good obfuscator either hides σ with respect to $\mathcal{A}$ or makes $\mathcal{A}$ harder!

Definitions: Potency

Let $\mathfrak{O} : \mathbb{P} \longrightarrow \mathbb{P}$ be an obfuscation. $\mathfrak{O}(P)$ is the obfuscation of $P \in \mathbb{P}$

Let $\{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ be a set of program analysis for the language $\mathbb{P}$:

$\mathfrak{O}$ is *potent* for P if

- ✔ $\exists \mathcal{A}_i \in \{\mathcal{A}_1, \ldots, \mathcal{A}_n\}$ such that $\mathfrak{O}$ is effective for P and $\mathcal{A}_i$;
- ✔ $\forall \mathcal{A}_j \in \{\mathcal{A}_1, \ldots, \mathcal{A}_n\} \wedge i \neq j$ $\mathfrak{O}$ is not defective for P.

Effectiveness deals with: precision and complexity!!

giaco CONFIDENTIAL

giaco CONFIDENTIAL

Measures of potency

Typical metric measures

- Program length $\mathfrak{O}(P)$ increases the operators and operands in P
- Cyclomatic complexity $\mathfrak{O}(P)$ increases the numer of predicates in P
- Nesting complexity $\mathfrak{O}(P)$ increases the nesting level of conditionals in P
- Data-flow complexity $\mathfrak{O}(P)$ increases the inter-block reference
- Fan-in/Fan-out complexity $\mathfrak{O}(P)$ increases the number of formal parameter and data structured referenced by P
- Data-structure complexity $\mathfrak{O}(P)$ increases the complexity of static data structures (arrays and records) declared in P
- OO metric $\mathfrak{O}(P)$ increases the number of methods, the depth in inheritance tree, the number of sub classes, and methods that can be generated in response to a message sent to an object

Is there a more general notion of potency?

ABSTRACT INTERPRETATION

[Cousot & Cousot '79]

A program P and a domain of computation for P: C

Semantic specification (interpreter): $[\![P]\!] : C \longrightarrow C$

(Approximate) observable properties: $\rho \in uco(C)$

DERIVE A SOUND APPROXIMATE SPECIFICATION $[\![P]\!]^\sharp$

$$\rho([\![P]\!](x)) \leq [\![P]\!]^\sharp(x)$$

THE LIMIT CASE: COMPLETENESS

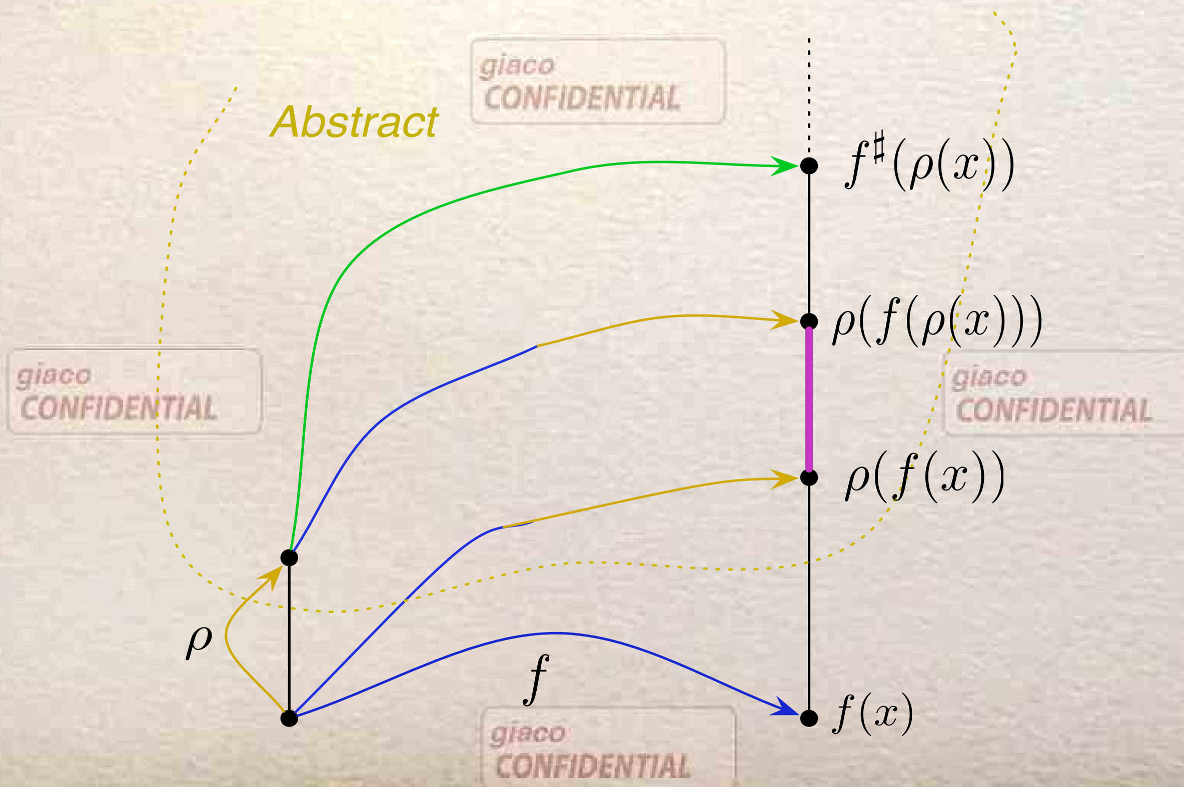
$$\rho([\![P]\!](x)) = [\![P]\!]^\sharp(x) \text{ iff } \rho([\![P]\!](x)) = \rho([\![P]\!](\rho(x)))$$

giaco CONFIDENTIAL

Completeness in Abstract Interpretation

Backward Soundness:
no information is lost by approximating the input/output

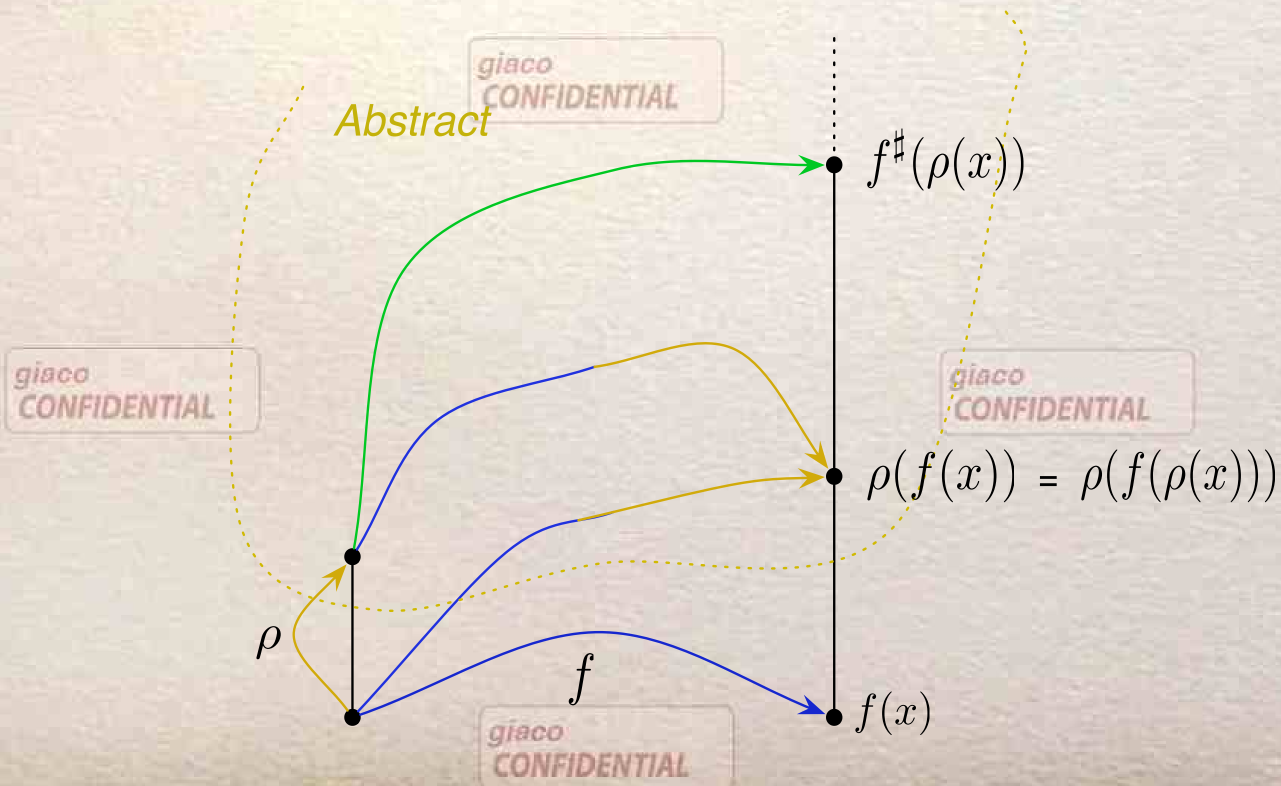
$\rho \circ f \leq \rho \circ f \circ \rho$



giaco CONFIDENTIAL

Completeness in Abstract Interpretation

- Backward Completeness:
  no loss of precision is accumulated by approximating the input
- $\rho \circ f = \rho \circ f \circ \rho$

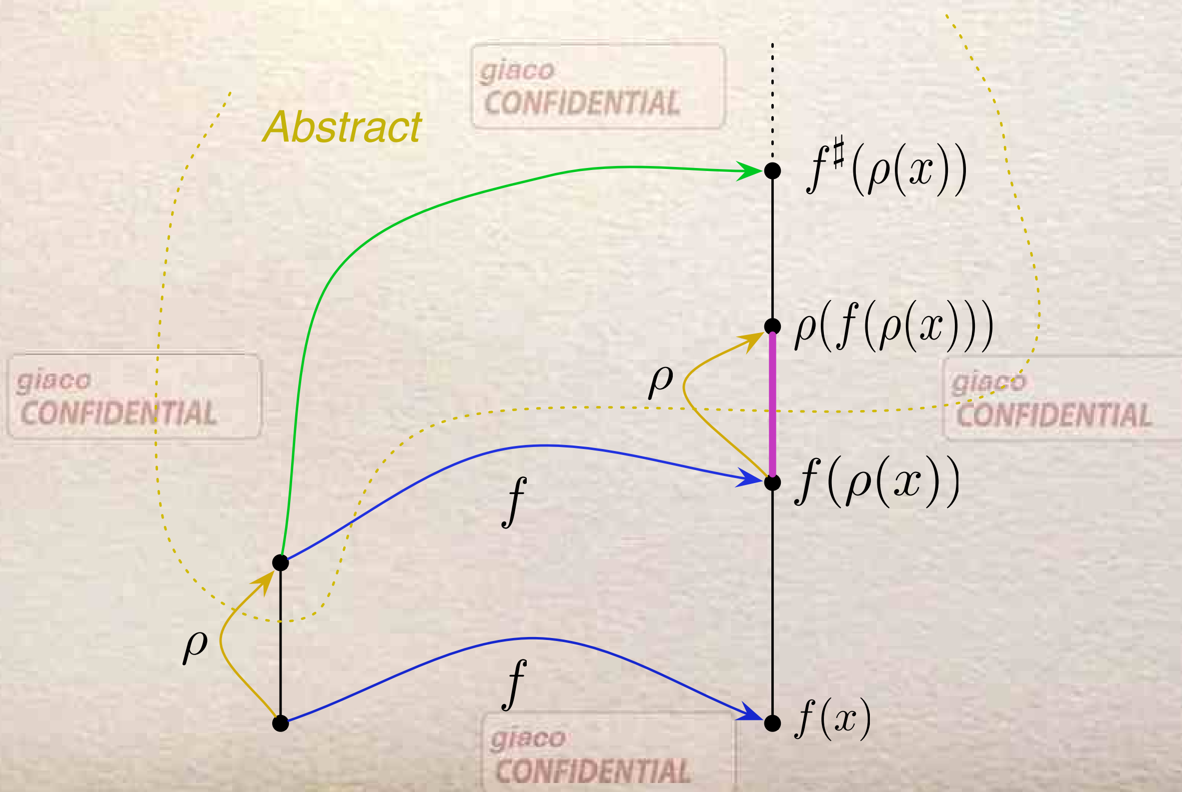


giaco CONFIDENTIAL

Completeness in Abstract Interpretation

Forward Completeness:
no information is lost by approximating the output

$f \circ \rho \leq \rho \circ f \circ \rho$

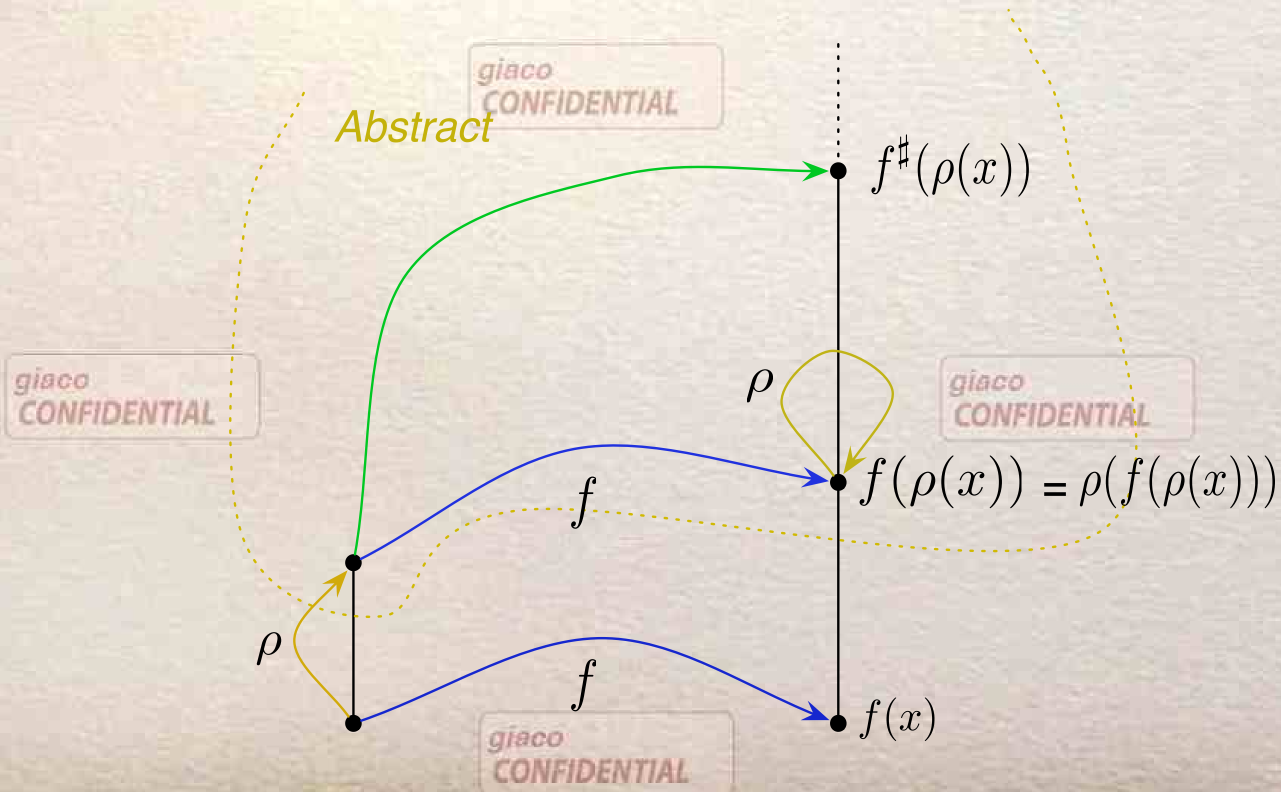


giaco CONFIDENTIAL

Completeness in Abstract Interpretation

Forward Completeness:
no information is lost by approximating the output

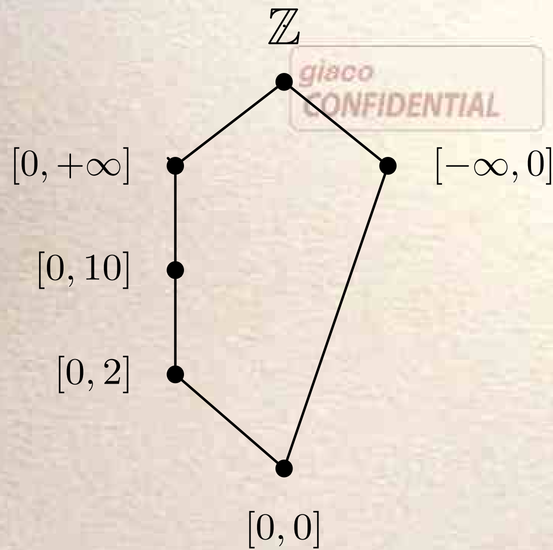
$f \circ \rho = \rho \circ f \circ \rho$



giaco CONFIDENTIAL

A CLASSICAL EXAMPLE

A SIMPLE EXAMPLE IN INTERVAL ANALYSIS

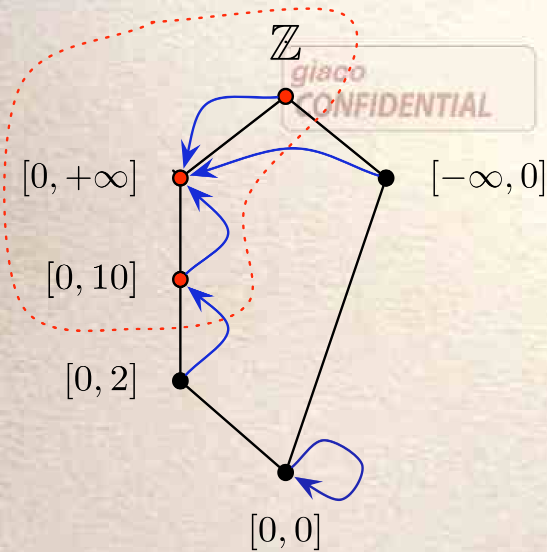


A simple domain of intervals

giaco CONFIDENTIAL

A CLASSICAL EXAMPLE

A SIMPLE EXAMPLE IN INTERVAL ANALYSIS



A simple domain of intervals

$$sq(X) = \left\{ \, x^2 \;\middle|\; x \in X \, \right\}$$

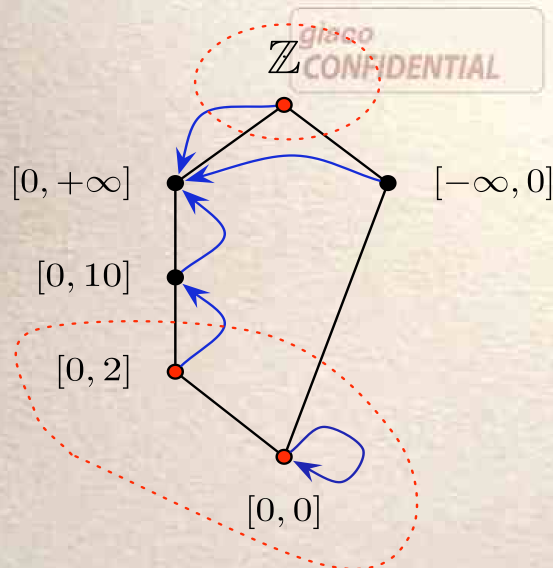
$\{\mathbb{Z}, [0, +\infty], [0, 10]\}$ is Forward but not Backward complete

giaco CONFIDENTIAL

giaco CONFIDENTIAL

A CLASSICAL EXAMPLE

A SIMPLE EXAMPLE IN INTERVAL ANALYSIS



A simple domain of intervals

$$sq(X) = \left\{ x^2 \,\middle|\, x \in X \right\}$$

$\{\mathbb{Z}, [0, +\infty], [0, 10]\}$ is Forward but not Backward complete

$\{\mathbb{Z}, [0, 2], [0, 0]\}$ is Backward but not Forward complete

OBSCURITY BY INCOMPLETENESS

Failing precision means failing completeness!

Obfuscating programs is making abstract interpreters incomplete

Let $\rho \in uco(\Sigma)$ with Σ semantic objects (data, traces etc)

A program transformation $\tau : \mathbb{P} \to \mathbb{P}$: $[\![P]\!] = [\![\tau(P)]\!]$.

ρ $\mathcal{B}$-complete for $[\![\cdot]\!]$ if $\rho([\![P]\!]) = [\![P]\!]^{\rho}$

τ obfuscates P if $[\![P]\!]^{\rho} \sqsubset [\![\tau(P)]\!]^{\rho}$

$$[\![P]\!]^{\rho} \sqsubset [\![\tau(P)]\!]^{\rho} \iff \rho([\![\tau(P)]\!]) \sqsubset [\![\tau(P)]\!]^{\rho}$$

giaco CONFIDENTIAL

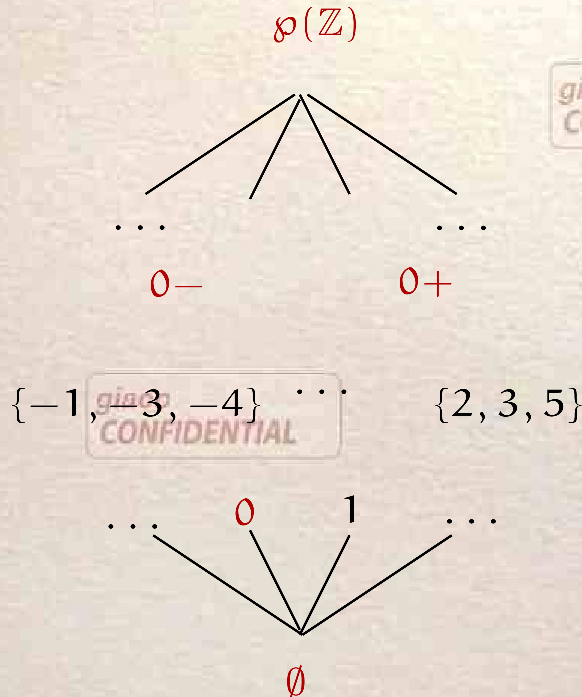
OBSCURITY BY INCOMPLETENESS

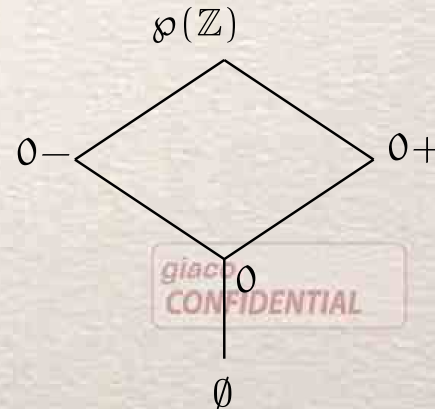
Failing precision means failing completeness!

Obfuscating programs is making abstract interpreters incomplete

$P : \mathtt{x} = \mathtt{a} * \mathtt{b}$

$Sign$ is an obvious abstraction of $\wp(\mathbb{Z})$:





giaco CONFIDENTIAL

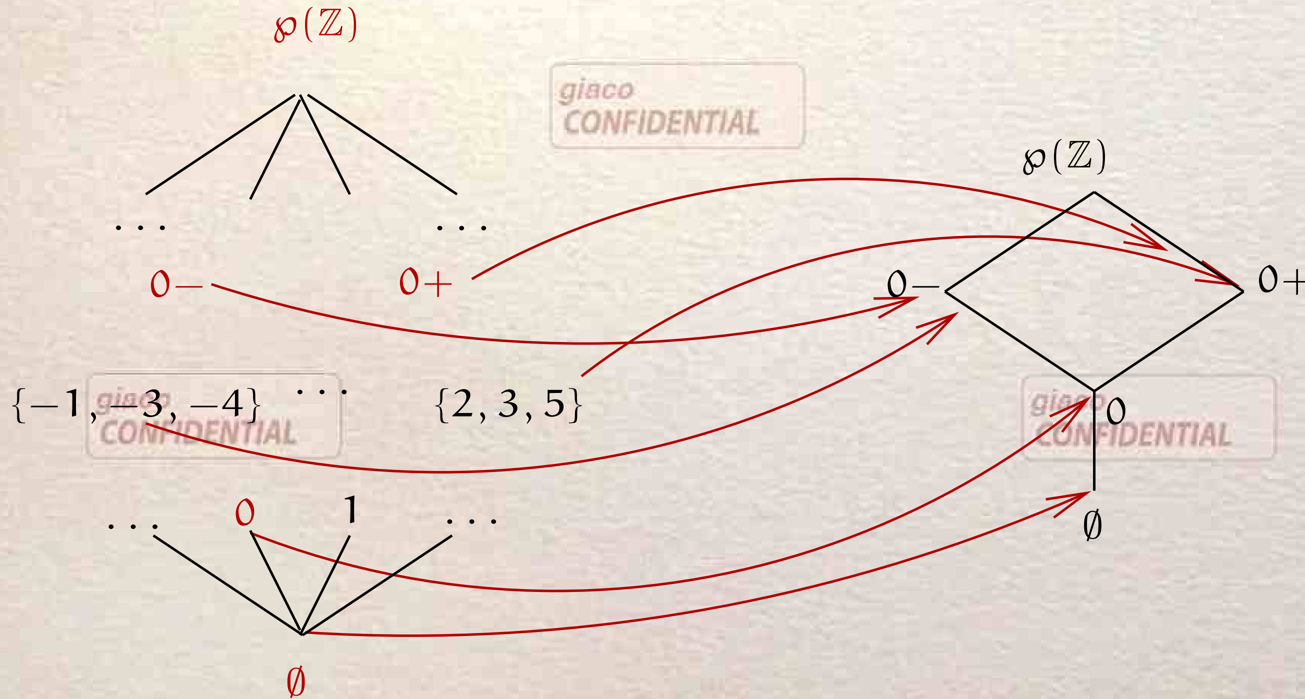
Obscurity by incompleteness

Failing precision means failing completeness!

Obfuscating programs is making abstract interpreters incomplete

$P : \mathtt{x} = \mathtt{a} * \mathtt{b}$

$Sign$ is an abstraction of $\wp(\mathbb{Z})$:



giaco CONFIDENTIAL

giaco CONFIDENTIAL

OBSCURITY BY INCOMPLETENESS

Failing precision means failing completeness!

Obfuscating programs is making abstract interpreters incomplete

$$P: \quad \mathtt{x} = \mathtt{a} * \mathtt{b} \quad \longrightarrow \quad \tau(P): \begin{array}{l} \mathtt{x} = 0; \\ \mathtt{if}\ \mathtt{b} \leq 0\ \mathtt{then}\ \{\mathtt{a} = -\mathtt{a}; \mathtt{b} = -\mathtt{b}\}; \\ \mathtt{while}\ \mathtt{b} \neq 0\ \{\mathtt{x} = \mathtt{a} + \mathtt{x}; \mathtt{b} = \mathtt{b} - 1\} \end{array}$$

$Sign$ is complete for P:

- ✔ $[\![P]\!]^{Sign} = \lambda \mathtt{a}, \mathtt{b}.\ Sign(\mathtt{a} * \mathtt{b})$

$Sign$ is incomplete for $\tau(P)$:

- ✔ $[\![\tau(P)]\!]^{Sign} = \lambda \mathtt{a}, \mathtt{b}. \begin{cases} 0 & \text{if } \mathtt{a} = 0 \vee \mathtt{b} = 0 \\ \wp(\mathbb{Z}) & \text{otherwise} \end{cases}$

Is there any way to get $\tau(P)$ systematically out of P?

Generalizing data-refinement I

We consider variable splitting:

$v \in Var(P)$ is split into $\langle v_1, v_2 \rangle$ such that
$v_1 = f_1(v)$, $v_2 = f_2(v)$ and $v = g(v_1, v_2)$

$$f_1(v) = v \div 10$$
$$f_2(v) = v \mod 10$$
$$g(v_1, v_2) = 10 \cdot v_1 + v_2$$

And the interval analysis: $\iota(x) = [\text{min}(x), \text{max}(x)]$

$$P : \left[\begin{array}{l} v = 0; \\ \textbf{while } v < N \ \{v++\} \end{array} \right. \qquad [\![P]\!]^{\iota} = \lambda v.\ [0, N]$$

giaco CONFIDENTIAL

Generalizing data-refinement I

We consider variable splitting:

$v \in Var(P)$ is split into $\langle v_1, v_2 \rangle$ such that
$v_1 = f_1(v)$, $v_2 = f_2(v)$ and $v = g(v_1, v_2)$

$$f_1(v) = v \div 10$$
$$f_2(v) = v \mod 10$$
$$g(v_1, v_2) = 10 \cdot v_1 + v_2$$

And the interval analysis: $\iota(x) = [\min(x), \max(x)]$

$$\tau(P): \left[\begin{array}{l} v_1 = 0; \\ v_2 = 0; \\ \textbf{while} \quad 10 \cdot v_1 + v_2 < N \; \{ \\ \qquad v_1 = v_1 + (v_2 + 1) \div 10 \\ \qquad v_2 = (v_2 + 1) \mod 10 \\ \}; \end{array}\right.$$

$$c: \quad v = 10 \cdot v_1 + v_2$$

$$\begin{array}{ll} [\![\tau(P); c]\!]^{\iota} & = \\ \lambda v.\, 10 \odot [0, \frac{N \ominus [0,9]}{10}] \oplus [0, 9] & = \\ \lambda v.\, [0, N] \oplus [0, 9] & = \\ \lambda v.\, [0, N+9] & \end{array}$$

Obfuscation induces errors

giaco CONFIDENTIAL

giaco CONFIDENTIAL

Generalising data-refinement II

We consider array splitting for weakening the invariant of Fibonacci's

$$\mathbf{Inv} = 2 \leq i \leq N \wedge \forall j \in [2, i].\ a[j] = a[j-1] + a[j-2]$$

The invariant **Inv** can be generated by relational interval-**Fib** analysis

$\eta = \alpha^+ \circ \alpha$ where

$$\alpha(X) = \begin{cases} \mathbf{Fib} & \text{if } \forall \langle S, x\rangle \in X.\ S \subseteq D_x \wedge \quad (S = \{0\} \wedge x[0] = 0) \vee \\ & \quad (S = \{0, 1\} \wedge x[0] = 0 \wedge x[1] = 1) \vee \\ & \quad (\forall j \in S.\ x[j] = x[j-1] + x[j-2]) \\ \mathbf{Any} & \text{otherwise} \end{cases}$$

$I \longrightarrow \mathbf{Fib}$ represents Fibonacci's sequences until $\max(I)$

$I \longrightarrow \mathbf{Any}$ represents any array with domain including I (no overlow)

$[n, m] \longrightarrow \mathbf{Fib} = [n, m-1] \longrightarrow \mathbf{Fib} \oplus [n, m-2] \longrightarrow \mathbf{Fib}$

GENERALISING DATA-REFINEMENT II

We consider array splitting for weakening the invariant of Fibonacci's

$$\mathbf{Inv} = 2 \leq i \leq N \wedge \forall j \in [2, i].\ a[j] = a[j-1] + a[j-2]$$

$$P : \left[\begin{array}{l} a[0] = 0; \\ a[1] = 1; \\ i = 2; \\ \textbf{while} \quad i \leq N \ \{ \\ \qquad a[i] = a[i-1] + a[i-2]; \\ \qquad i++ \\ \quad \} \end{array}\right.$$

$$[\![P]\!]^{\mathfrak{u} \longrightarrow \eta} = a \in [0, N] \longrightarrow \mathbf{Fib} \wedge i \in [2, N+1]$$

giaco CONFIDENTIAL

Generalising data-refinement II

We consider array splitting for weakening the invariant of Fibonacci's

$$\textbf{Inv} = 2 \leq i \leq N \wedge \forall j \in [2, i].\ a[j] = a[j-1] + a[j-2]$$

$$\tau(P):\left[\begin{array}{l} b[0] = 0; \\ c[0] = 1; \\ i = 2; \\ \textbf{while}\quad i \leq N\ \{ \\ \quad\quad \textbf{if}\quad i \mod 2 == 0 \\ \quad\quad\quad \{b[i \div 2] = c[(i-1) \div 2] + b[(i-2) \div 2]\} \\ \quad\quad\quad \{c[i \div 2] = b[(i) \div 2] + c[(i-2) \div 2]\}; \\ \quad\quad\quad\ i++ \\ \quad\quad\quad \} \end{array}\right.$$

$$[\![\tau(P)]\!]^{\mathfrak{u} \longrightarrow \eta} = b, c \in [0, N \div 2] \longrightarrow \textbf{Any} \wedge i \in [2, N+1]$$

giaco CONFIDENTIAL

Generalising data-refinement II

We consider array splitting for weakening the invariant of Fibonacci's

$$\mathbf{Inv} = 2 \leq i \leq N \wedge \forall j \in [2, i].\ a[j] = a[j-1] + a[j-2]$$

The analysis of $\tau(P)$ is unable to get $\mathbf{Inv}$ back!!

giaco CONFIDENTIAL

STANDARD OBFUSCATION

1. Abstraction transformations: Destroy module structure, classes, functions, etc.!
2. Data transformations: Replace data structures with new representations!
3. Control transformations: Destroy if-, while-, repeat-, etc.!
4. Dynamic transformations: Make the program change at runtime!

giaco CONFIDENTIAL

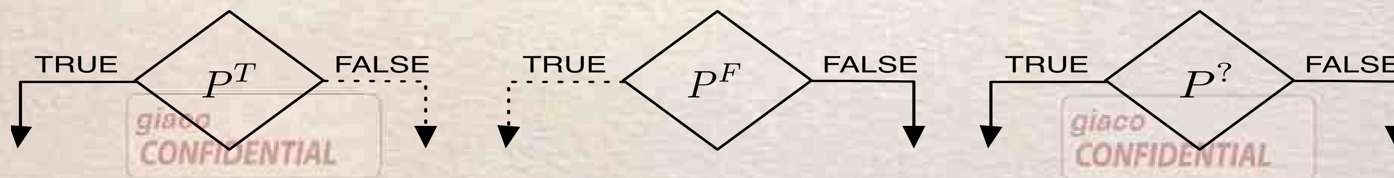
Opaque predicates

Simply put:

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

Notation:

- ✔ P^T for an opaquely true predicate
- ✔ P^F for an opaquely false predicate
- ✔ $P^?$ for an unknown predicate



giaco CONFIDENTIAL

Opaque predicates

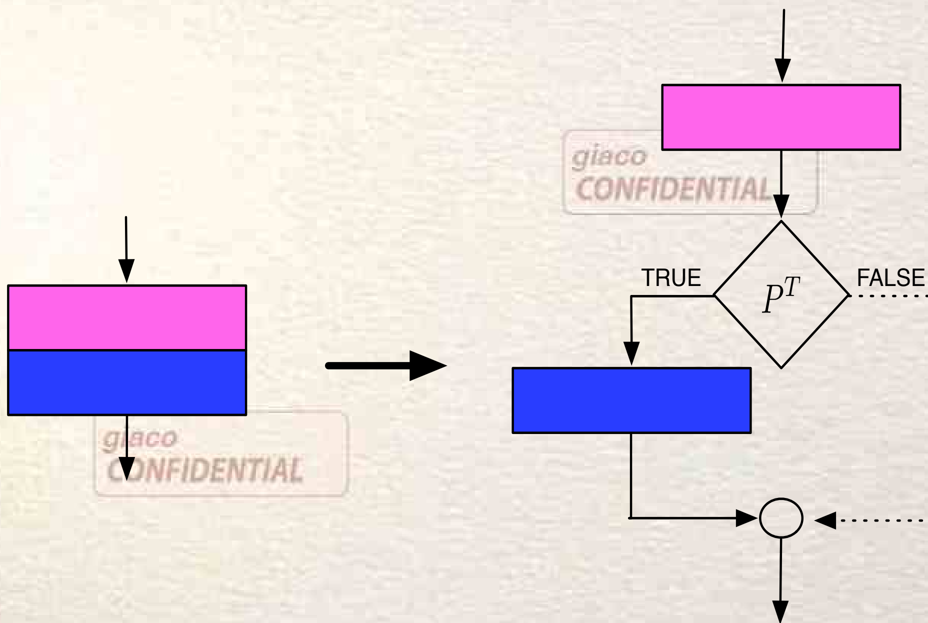
Examples of opaque predicates from number theory

$$
\begin{aligned}
\forall x, y \in \mathbb{Z} :&\quad 7y^2 - 1 \neq x^2 \\
\forall x \in \mathbb{Z} :&\quad 2 \mid (x + x^2) \\
\forall x \in \mathbb{Z} :&\quad 3 \mid (x^3 - x) \\
\forall n \in \mathbb{Z}^+, x, y \in \mathbb{Z} :&\quad (x - y) \mid (x^n - y^n) \\
\forall n \in \mathbb{Z}^+, x, y \in \mathbb{Z} :&\quad 2 \mid n \vee (x + y) \mid (x^n + y^n) \\
\forall n \in \mathbb{Z}^+, x, y \in \mathbb{Z} :&\quad 2 \nmid n \vee (x + y) \mid (x^n - y^n) \\
\forall x \in \mathbb{Z}^+ :&\quad 9 \mid (10^x + 3 \cdot 4^{(x+2)} + 5) \\
\forall x \in \mathbb{Z} :&\quad 3 \mid (7x - 5) \Rightarrow 9 \mid (28x^2 - 13x - 5) \\
\forall x \in \mathbb{Z} :&\quad 5 \mid (2x - 1) \Rightarrow 25 \mid (14x^2 - 19x - 19) \\
\forall x, y, z \in \mathbb{Z} :&\quad (2 \nmid x \wedge 2 \nmid y) \Rightarrow x^2 + y^2 \neq z^2 \\
\forall x \in \mathbb{Z}^+ :&\quad 14 \mid (3 \cdot 7^{4x+2} + 5 \cdot 4^{2x-1} - 5)
\end{aligned}
$$

giaco CONFIDENTIAL

OPAQUE PREDICATE INSERTION

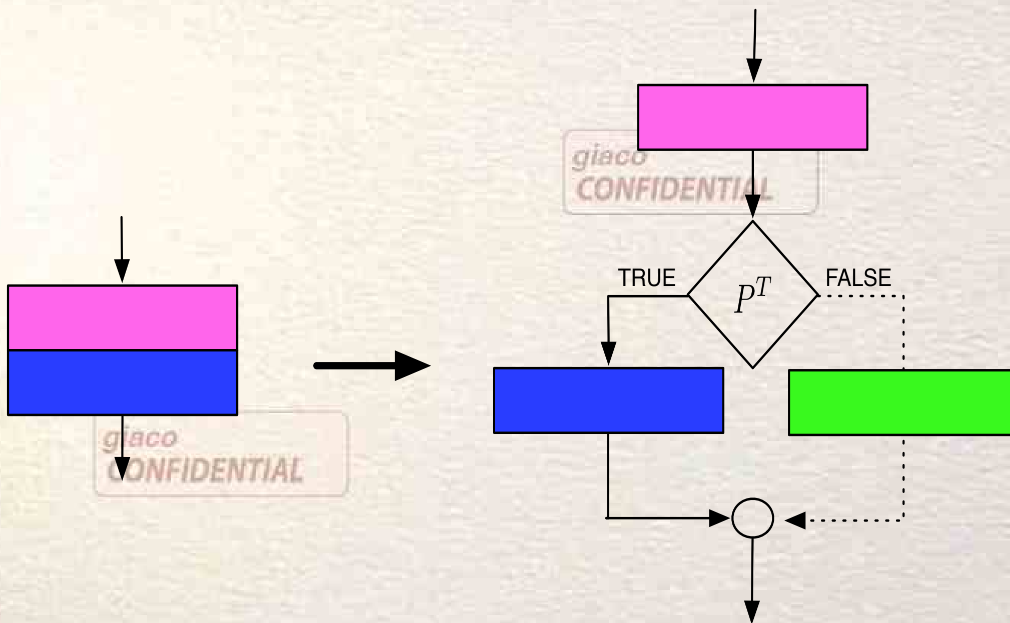
The simplest block splitting!



Attack:
Crack the predicate P^T

giaco CONFIDENTIAL

The green block can be bogous!

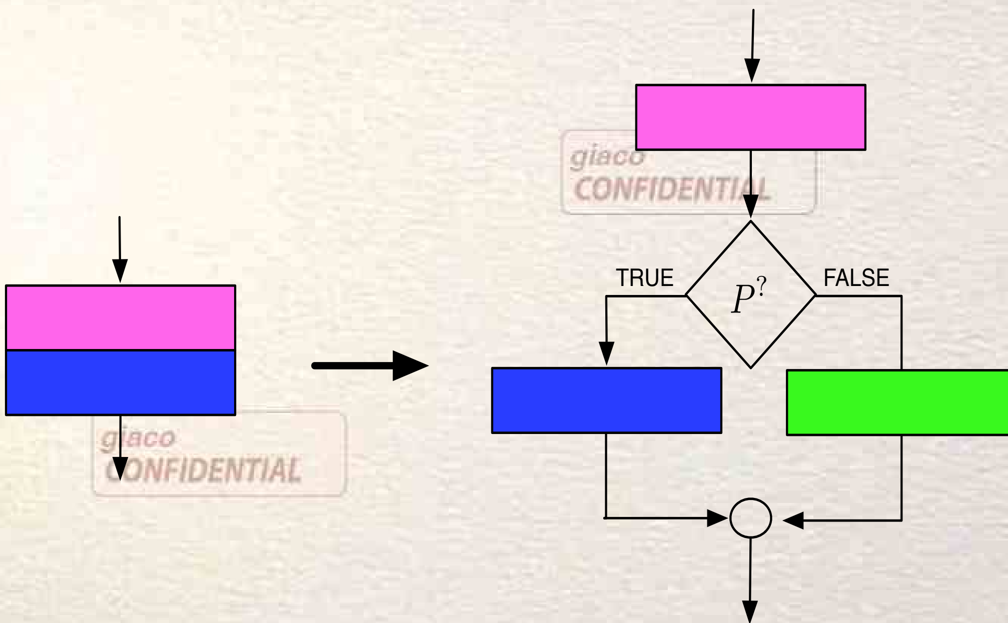


Attack:
Crack the predicate P^T

giaco CONFIDENTIAL

Opaque predicate insertion

The blue and green blocks should be semantically equivalent!

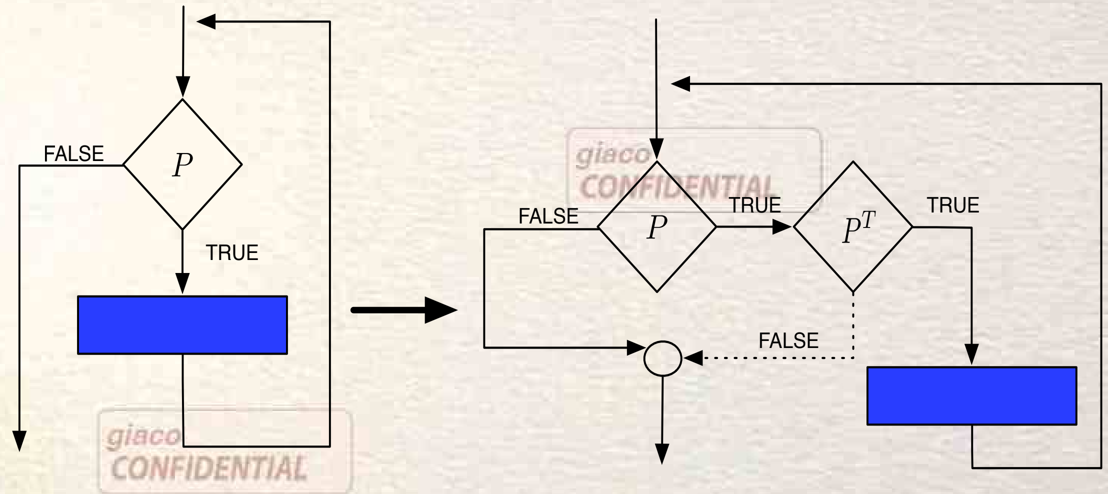


Attack:
Analyse semantic equivalence!

giaco CONFIDENTIAL

Opaque predicate insertion

By modifying loop conditions
P becomes $P \wedge P^T$

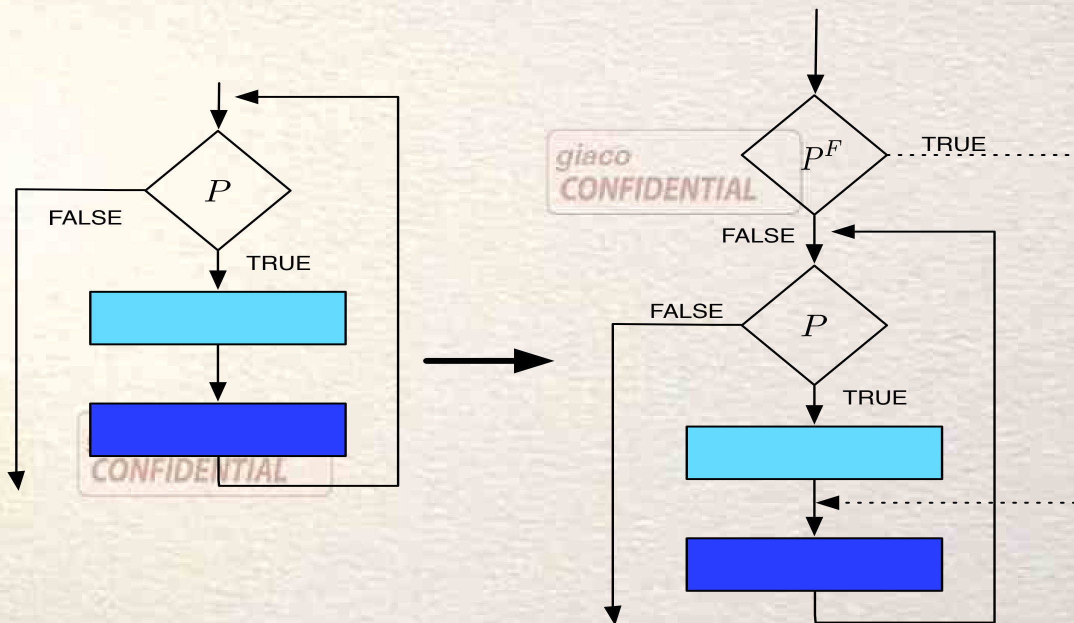


Attack:
Crack the predicate P^T

giaco CONFIDENTIAL

OPAQUE PREDICATE INSERTION

Making loops irreducible
by jumping into a loop!

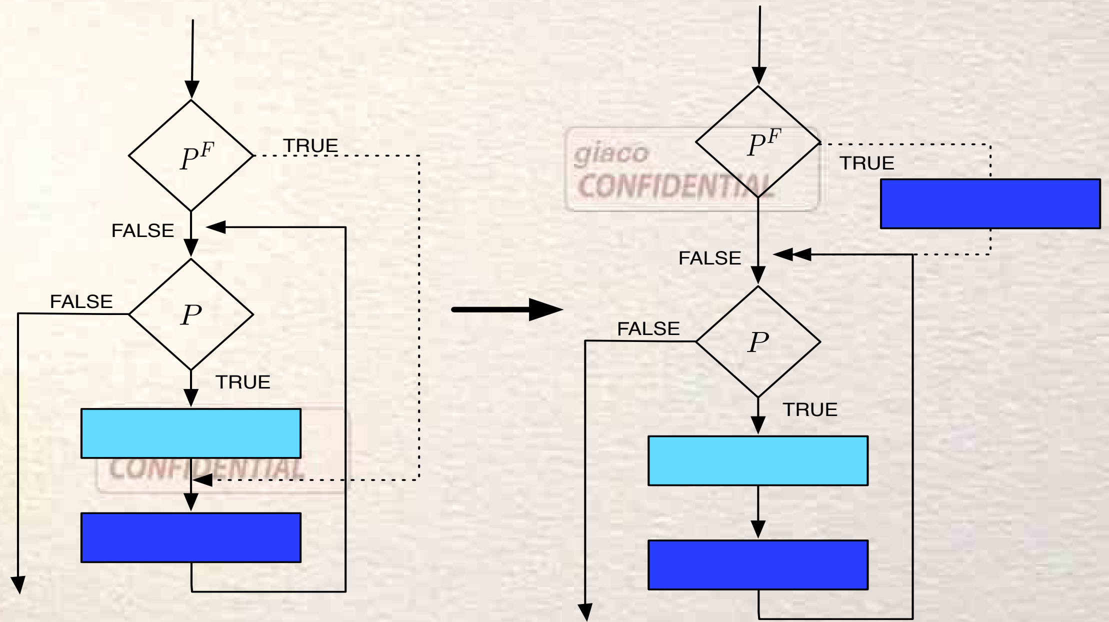


Attack:
Crack the predicate P^F

giaco CONFIDENTIAL

Opaque predicate attack

Making obfuscated loops
reducible
by *code replication*!



Attack:
Very hard to get due to exponential blowup!

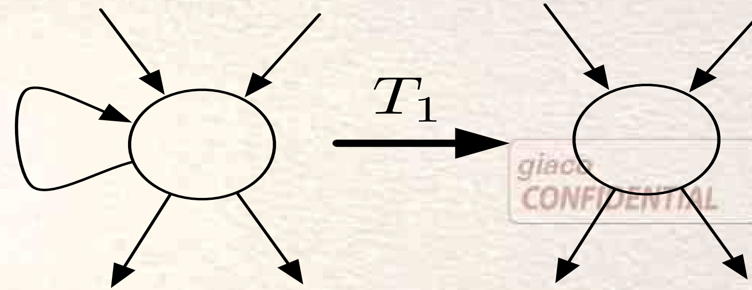
L. Carter, J. Ferrante, C. Thomborson

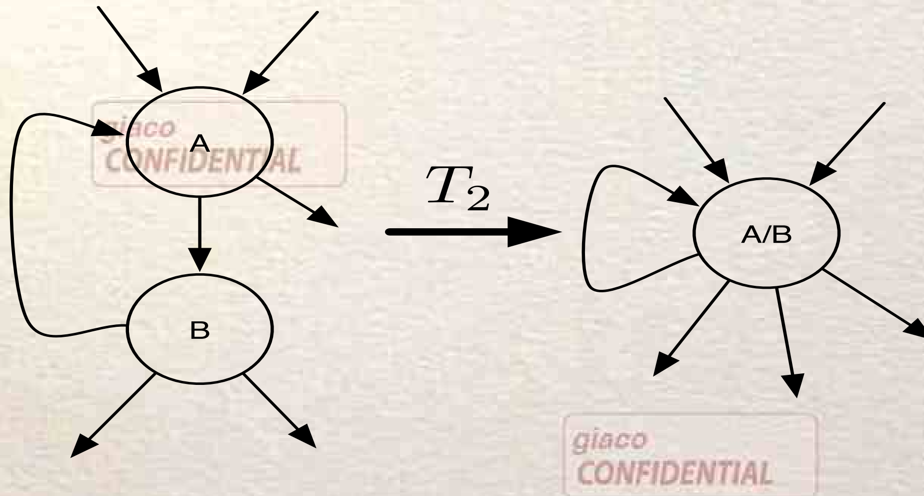
Folklore Confirmed: Reducible Flow Graphs are Exponentially Larger, POPL 2003

giaco CONFIDENTIAL

OPAQUE PREDICATE ATTACK

G is reducible if $G \overset{T_1/T_2}{\Longrightarrow}{}^{*} \odot$



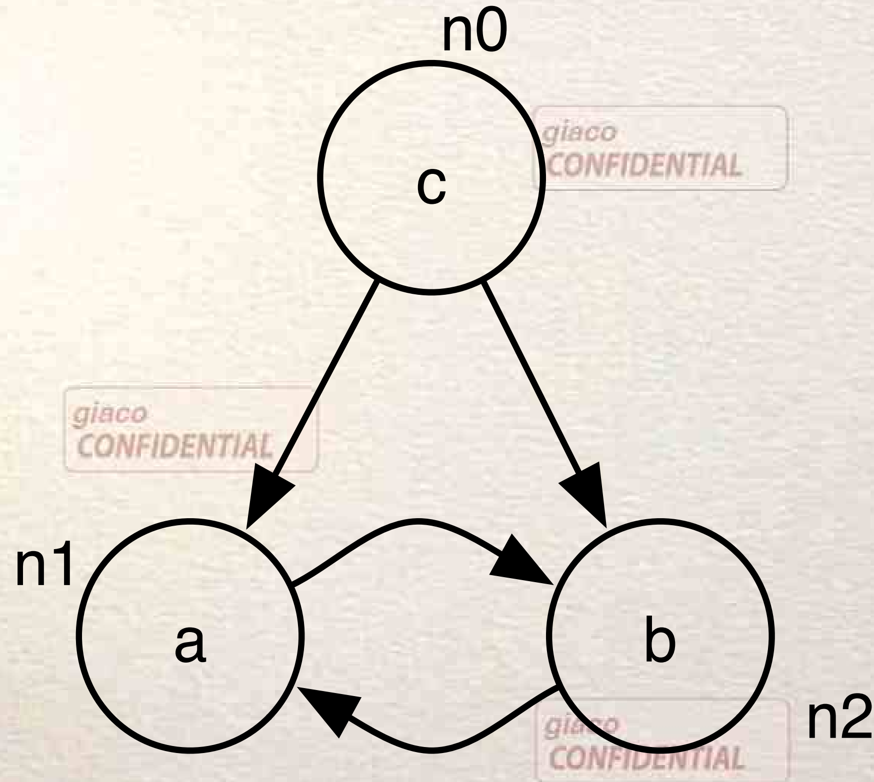


Attack:
Very hard to get due to exponential blowup!

giaco CONFIDENTIAL

Opaque predicate attack

```
c: if (cond) goto b
a: x = ...
b: y = ...
   goto a
```

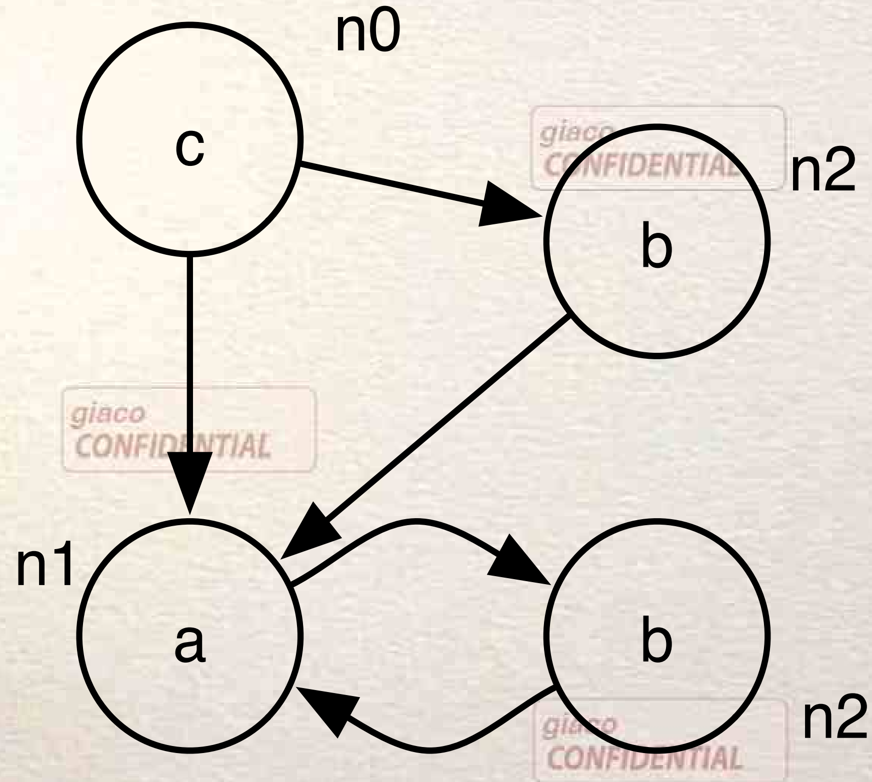


Attack:
Typical decompiler attack!

giaco CONFIDENTIAL

Opaque predicate attack

```
c: if (cond) then
       y = ...
   while (T) {
       x = ...
       y = ...
   }
```



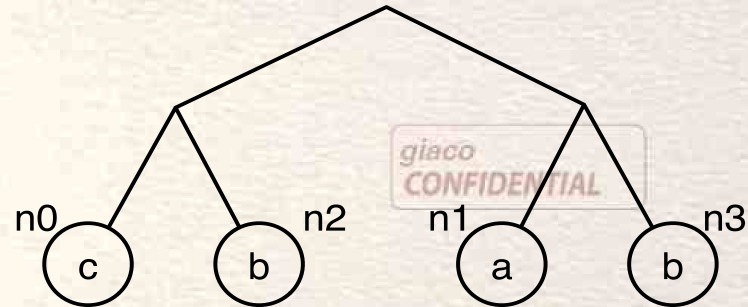
Attack:
Typical decompiler attack!

giaco CONFIDENTIAL

Opaque predicate attack

A leftist tree T is simply a rooted binary tree

$\pi = p_1 p_2 ... p_k$ is a *leftist leaf sequence* of T if $\exists T_i$ subtree of T such that $p_1 \in T_i$ and $p_{i+1} = left_l(T_i) \vee p_{i+1} = left_r(T_i)$



$n_0 n_2 n_2 n_1 n_0 n_1$ is ok
$n_0 n_3$ is wrong!

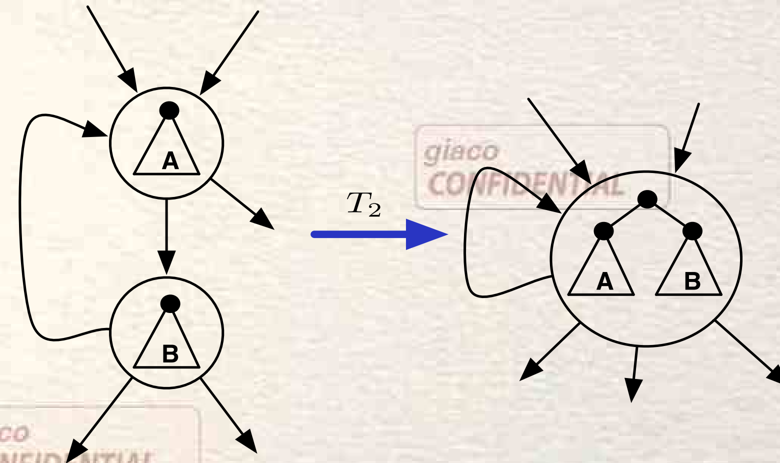
Idea:
Associate a leftist tree with reducible graphs!

Fact:
$\Gamma \subseteq \Sigma$, T is Γ-complete if $\forall \sigma \in \Gamma^* :$ σ is suffix of $\mathcal{L}(T)$
If $|\Gamma| = n$ the minimal number of leaves of a Γ-complete tree is 2^{n-1}

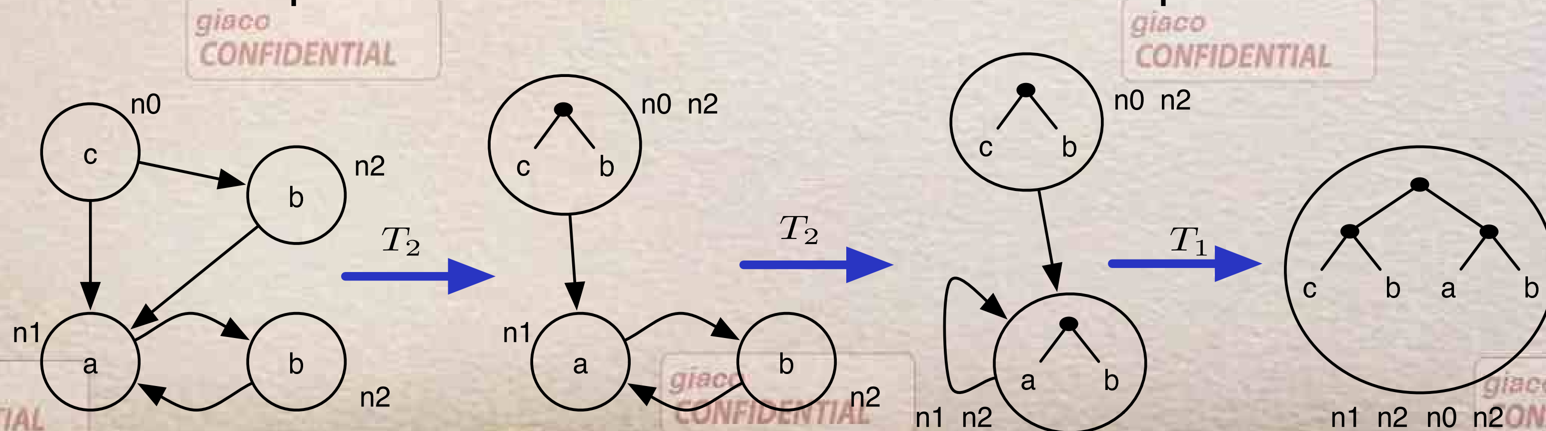
giaco CONFIDENTIAL

Exponential blowup

Theorem:
If R is a *reducible graph* equivalent to a graph with n nodes (with labels in Γ) with initial label c and $\mathcal{L}(R) = c\Gamma^*$, then R must have at least $2^{|\Gamma|-1}$ nodes.



Proof:
Associate a leftist tree with reducible graphs!
The tree corresponds to the basic transformations required for reducibility



giaco CONFIDENTIAL

Opaque predicates and exponential blowup

Note:

complete flow graph, have no equivalent reducible flow graph that is not exponential in size

Consequence:

- No node splitting technique can avoid this exponential blowup.
- Automatic compiler analyses would find handling such large programs difficult
- distribute code whose flow graph is irreducible!!

$\Longrightarrow$ exponential number of opaque predicates!!

L. Carter, J. Ferrante, C. Thomborson

Folklore Confirmed: Reducible Flow Graphs are Exponentially Larger, POPL 2003

---

giaco CONFIDENTIAL

Opaque predicates from pointer aliasing

Generate opaque predicates by spurious aliases

Potency measure by complexity of static analysis

1-level aliasing is easy **P** [Banning '79]

≥ 2-level aliasing is hard **NP** [Horowitz '97]

with dynamic memory allocation is undecidable!!

understanding control-flow = solve a ≥ 2-level aliasing problem

---

C. Collberg, C. Thomborson, D. Low

Manufacturing cheap, resilient, and stealthy opaque precicates, POPL 1998

---

giaco CONFIDENTIAL

Opaque predicates from pointer aliasing

`Input`: A program P

1. $P := P \cup R_b$ such that R builds a *dynamically allocated pointer-structure* $G = \{G_1, G_2, \ldots\}$
2. $P := P \cup R_p$ such that R_p add pointers $Q = \{q_1, q_2, \ldots\}$ points to structures in G
3. Construct a series of invariants $I = \{I_1, I_2, \ldots\}$ over G and Q such that
   - ✔ q_j always points to G_j
   - ✔ G_i is always strongly connected
4. $P := P \cup R_m$ such that R_m *occasionally* modifies the structures in G and pointers in Q while keeping invariants I
5. Using I construct opaque predicates over Q such that:
   - ✔ $(q_i \neq q_j)^T$ if q_i and q_j are known to point into different graphs
   - ✔ $(q_i \neq \texttt{null})^T$ if q_i is inside G_i and G_i has no leaf nodes
   - ✔ $(q_i = q_j)^?$ if q_i and q_j are known to both point into G_k

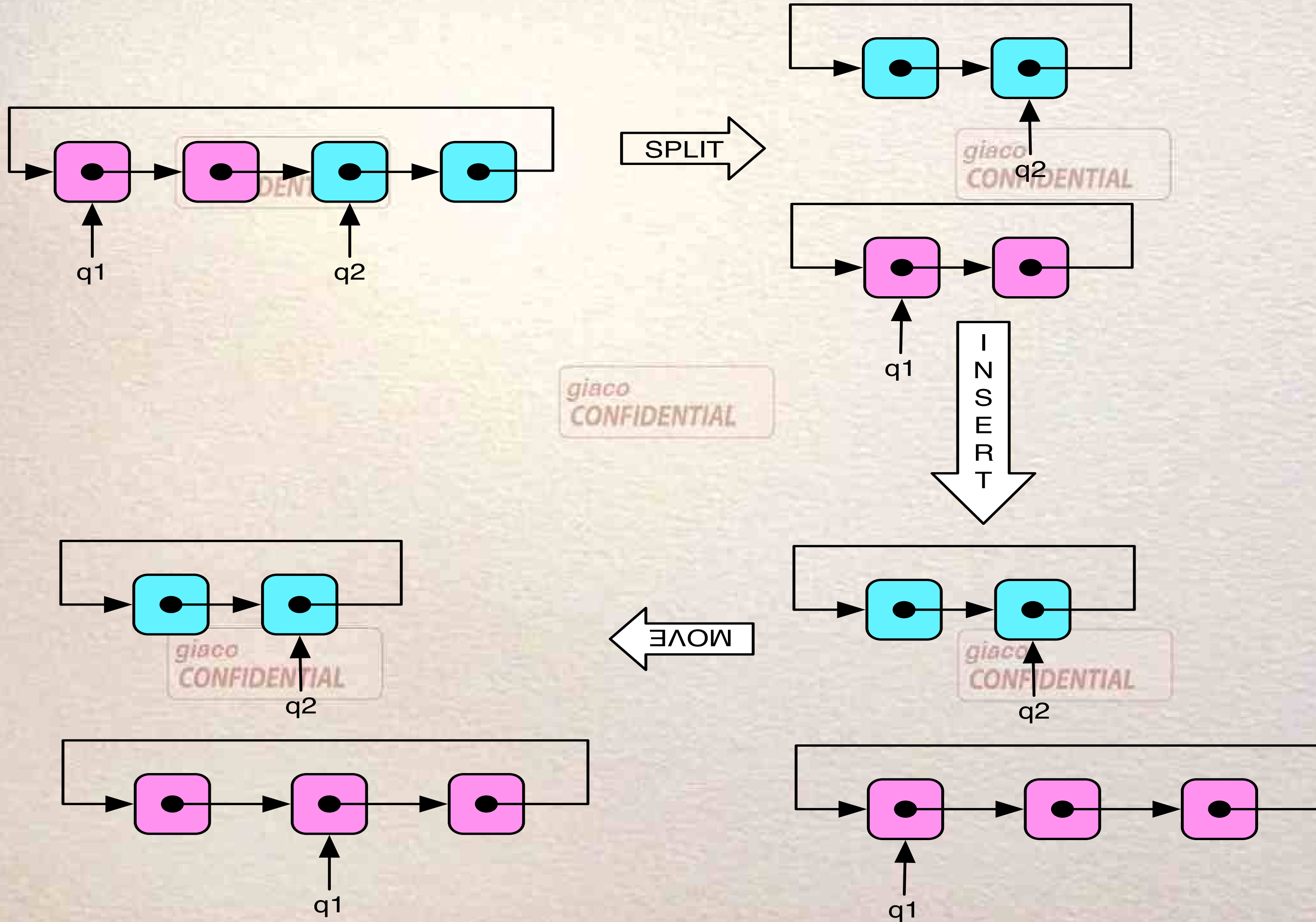
giaco CONFIDENTIAL

Opaque predicates from pointer aliasing

- Two invariants:
  1. G_1 and G_2 are circular linked lists
  2. q_1 points to a node in G_1 and q_2 points to a node in G_2.
- Perform enough operations to confuse even the most precise alias analysis algorithm
- Insert opaque queries such as $(q_1 \neq q_2)^T$ into the code.

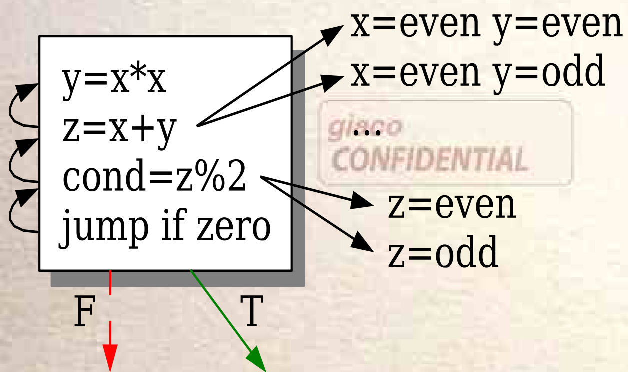
giaco CONFIDENTIAL

OPAQUE PREDICATES FROM POINTER ALIASING



giaco CONFIDENTIAL

A MEASURE OF POTENCY



Breaking $\forall x \in \mathbb{Z},\ 2|(x^2+x)$
in SPECint2000 Bench

	# opaque pred	time obf	time deobf
bzip2	442	0.53	0.13
crafty	3298	35.18	0.47
gap	6659	8.59	1.02
gcc	28006	476.33	3.11
gzip	734	0.44	0.11
mcf	189	0.29	0.06
parser	2543	2.48	0.31
perlbmk	10255	42.71	1.23
twolf	3575	1.88	0.48
vortex	8269	8.79	0.91
vpr	2206	1.44	0.3
Total	**66176**	**578.66**	**8.13**

The brute force attack took 8.83 seconds to detect only 1 opaque predicate

Example: $3 \mod (x^3 - x) = 0$

Abstract interpretation-based de-obfuscation attack took 8.13 seconds to de-obfuscate 66176 opaque predicates in a 32-bit env.

giaco CONFIDENTIAL

giaco CONFIDENTIAL

What we have learned so far?

Dalla Preda & Giacobazzi. Semantic-based Code Obfuscation by Abstract Interpretation. J. of Computer Security 2009

Attacking is refining the abstract interpreter towards completeness

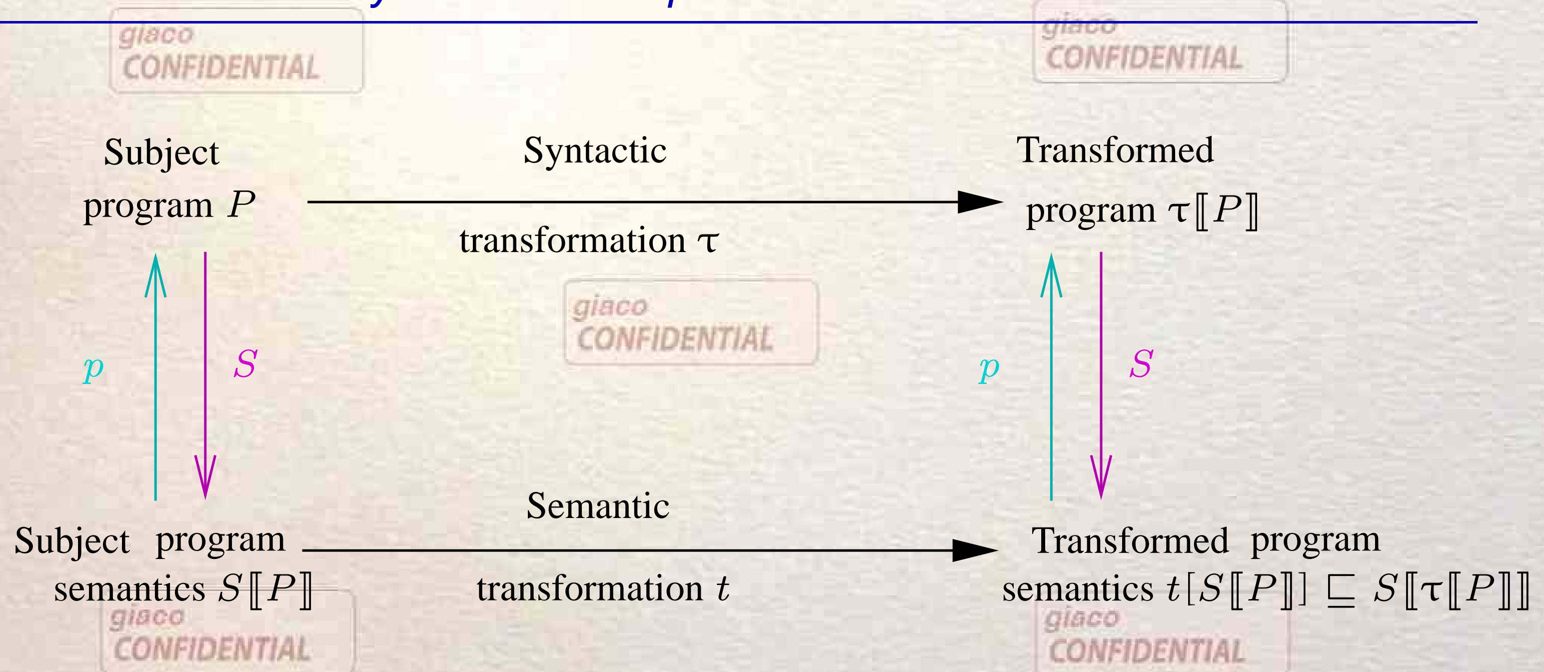
- A good measure of Potency
- We know how to do it!!!
  - By abstract domain refinement
  - By spurious counterexample elimination
  - By Dynamic analysis

Obfuscating is making (abstract) interpreters incomplete

- Is there any systematic way to do it?

OBFUSCATION BY PROGRAM TRANSFORMATION

Cousot & Cousot, Systematic Design of Program Transformation Frameworks by Abstract Interpretation. POPL'02



Syntactic transformation: $\tau = p \circ t \circ S$

giaco CONFIDENTIAL

The Geometry of Semantics Transformers

Making semantics complete (from above and below):

$$\mathbb{F}^{\uparrow}_{\eta,\rho}(f) = \bigsqcap\{h : C \longrightarrow C \mid f \sqsubseteq h,\ \rho \circ h \circ \eta = h \circ \eta\}$$
$$\mathbb{F}^{\downarrow}_{\eta,\rho}(f) = \bigsqcup\{h : C \longrightarrow C \mid f \sqsupseteq h,\ \rho \circ h \circ \eta = h \circ \eta\}$$

$\mathbb{F}^{\uparrow}_{\eta,\rho}(f)$ and $\mathbb{F}^{\downarrow}_{\eta,\rho}(f)$ are (Forward) complete

Making semantics maximally in-complete (from above and below):

$$\mathbb{O}^{\uparrow}_{\eta,\rho}(f) = \bigsqcup\{g : C \longrightarrow C \mid \mathbb{F}^{\downarrow}_{\eta,\rho}(g) = \mathbb{F}^{\downarrow}_{\eta,\rho}(f)\}$$
$$\mathbb{O}^{\downarrow}_{\eta,\rho}(f) = \bigsqcap\{g : C \longrightarrow C \mid \mathbb{F}^{\uparrow}_{\eta,\rho}(g) = \mathbb{F}^{\uparrow}_{\eta,\rho}(f)\}$$

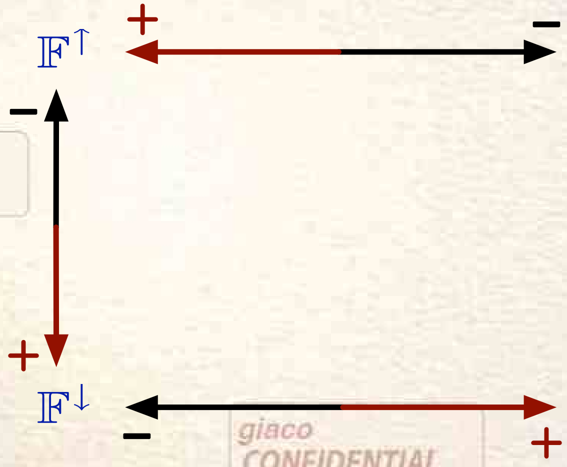
$\mathbb{O}^{\uparrow}_{\eta,\rho}(f)$ and $\mathbb{O}^{\downarrow}_{\eta,\rho}(f)$ are generally in-complete

giaco CONFIDENTIAL

The Geometry of Semantics Transformers

Minimal complete transformation from above

Maximal incomplete transformation from below



Minimal complete transformation from below

Maximal incomplete transformation from above

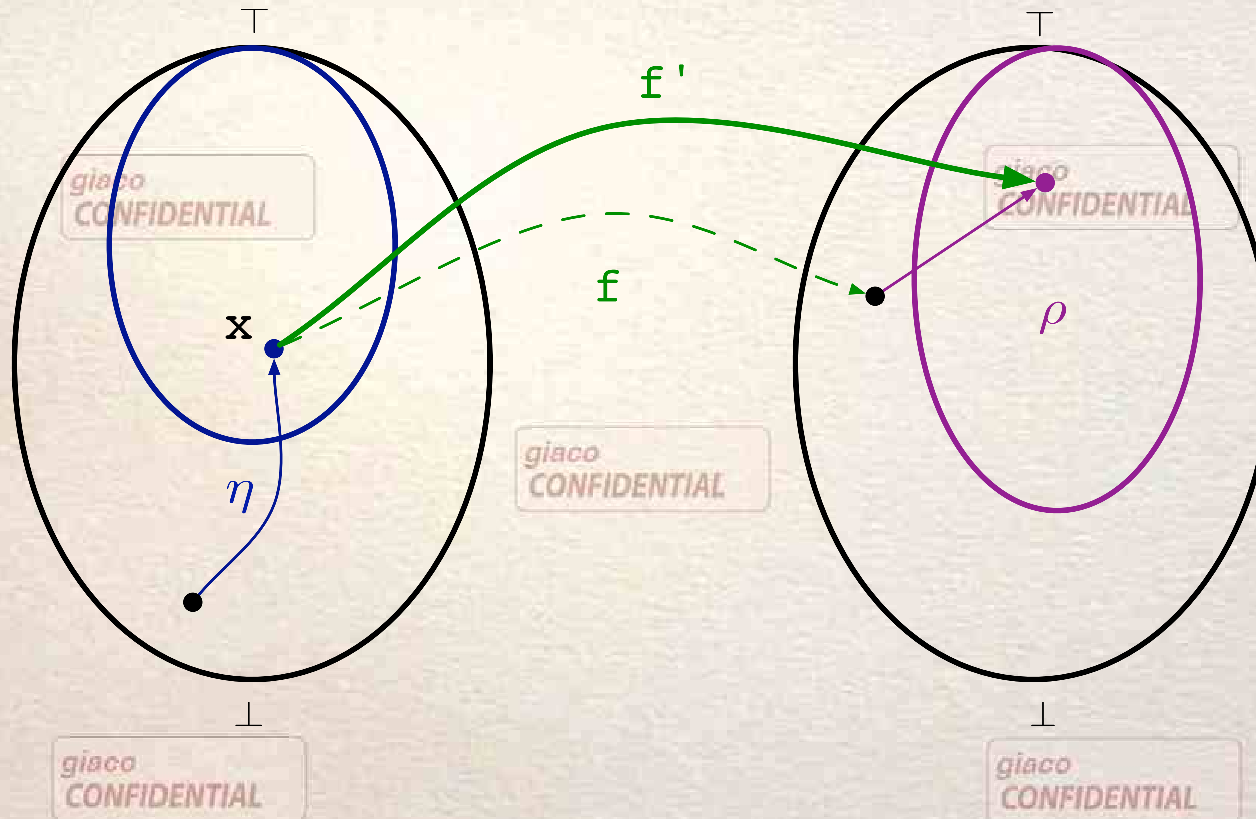
$$(\mathbb{F}^{\uparrow})^{+} = \mathbb{F}^{\downarrow} \quad \text{and} \quad (\mathbb{F}^{\uparrow})^{-} = \mathbb{O}^{\downarrow}$$

Giacobazzi & Mastroeni, Transforming abstract interpretations by abstract interpretation SAS'08

---

giaco CONFIDENTIAL

The Geometry of Semantics Transformers

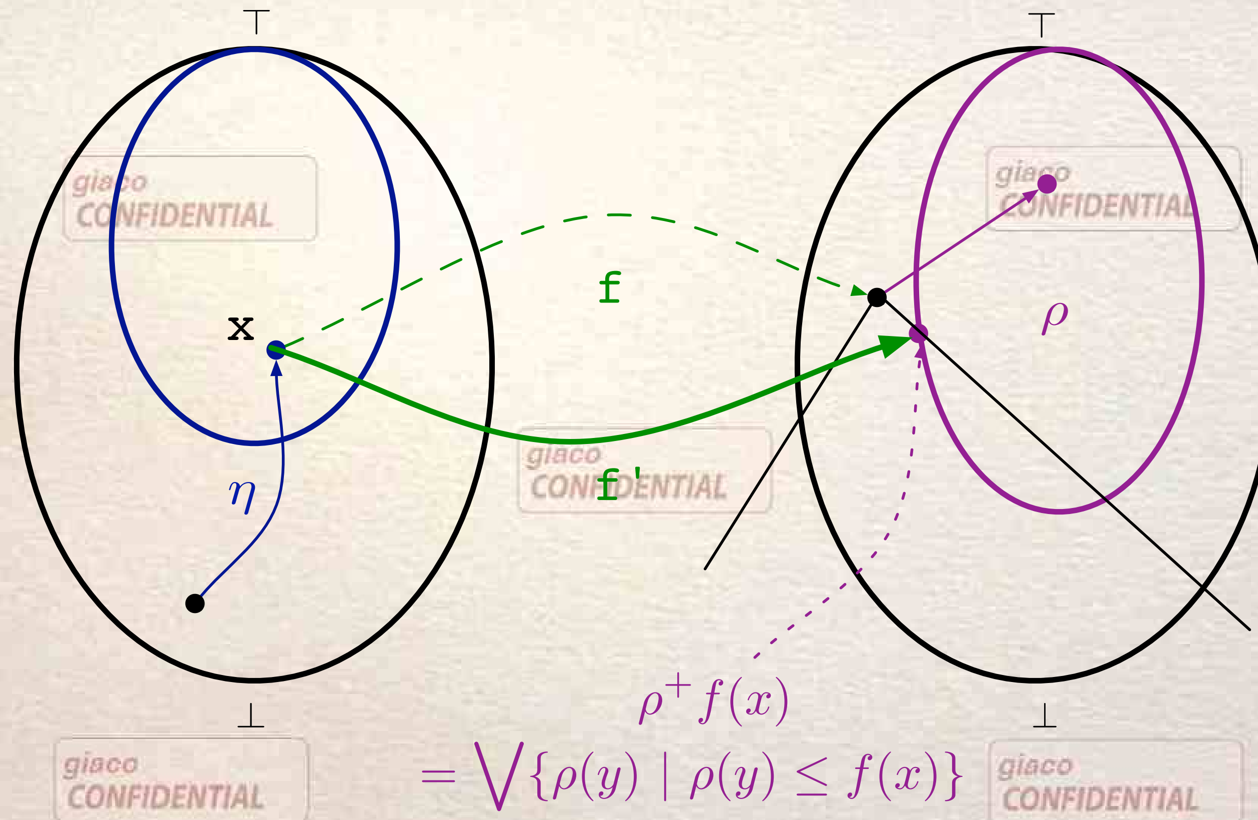


Making forward completeness: Transforming the semantics upwards

$$\mathbb{F}^{\uparrow}_{\eta,\rho} = \lambda f.\lambda x. \begin{cases} \rho \circ f(x) & \text{if } x \in \eta(C) \\ f(x) & \text{otherwise} \end{cases}$$

giaco CONFIDENTIAL

The Geometry of Semantics Transformers

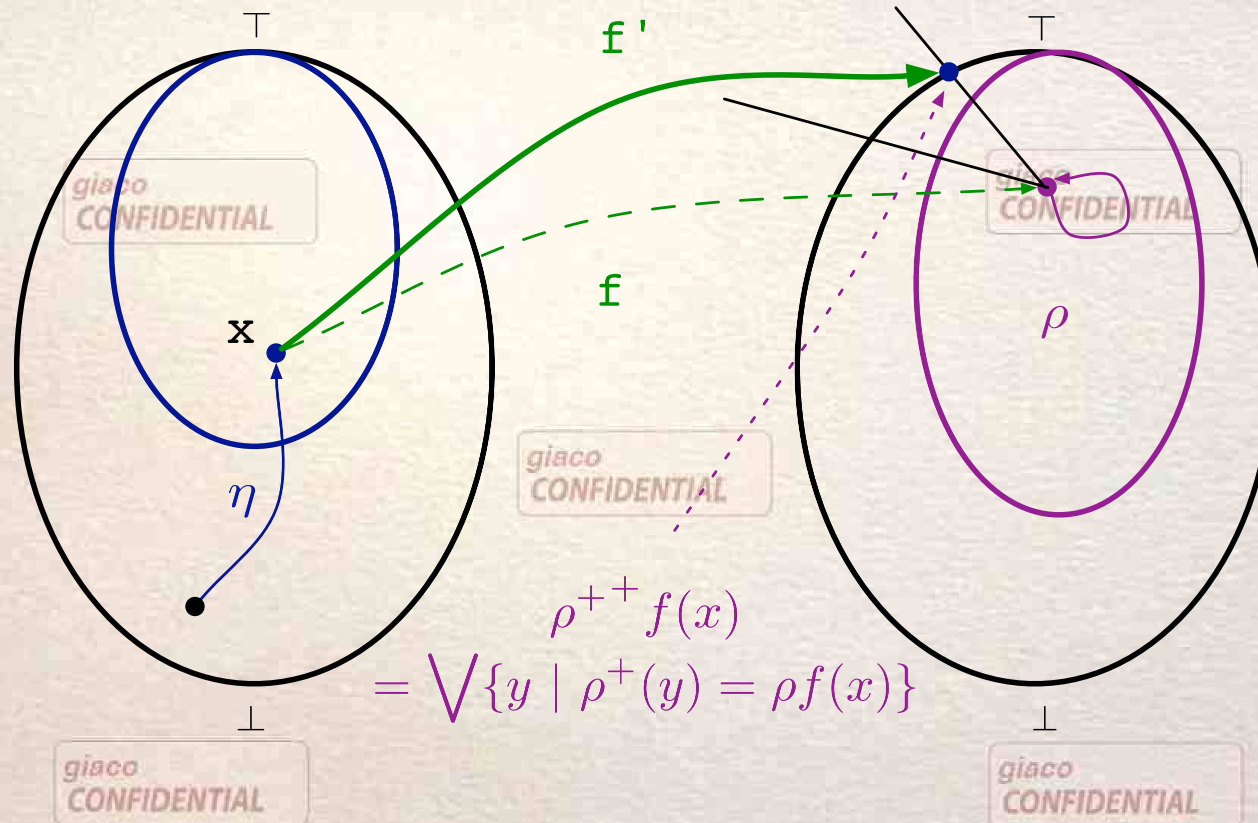


Making FORWARD COMPLETENESS: Transforming the semantics downwards

$$\mathbb{F}^{\downarrow}_{\eta,\rho} = \lambda f.\lambda x. \begin{cases} \rho^+ \circ f(x) & \text{if } x \in \eta(C) \\ f(x) & \text{otherwise} \end{cases}$$

giaco CONFIDENTIAL

THE GEOMETRY OF SEMANTICS TRANSFORMERS

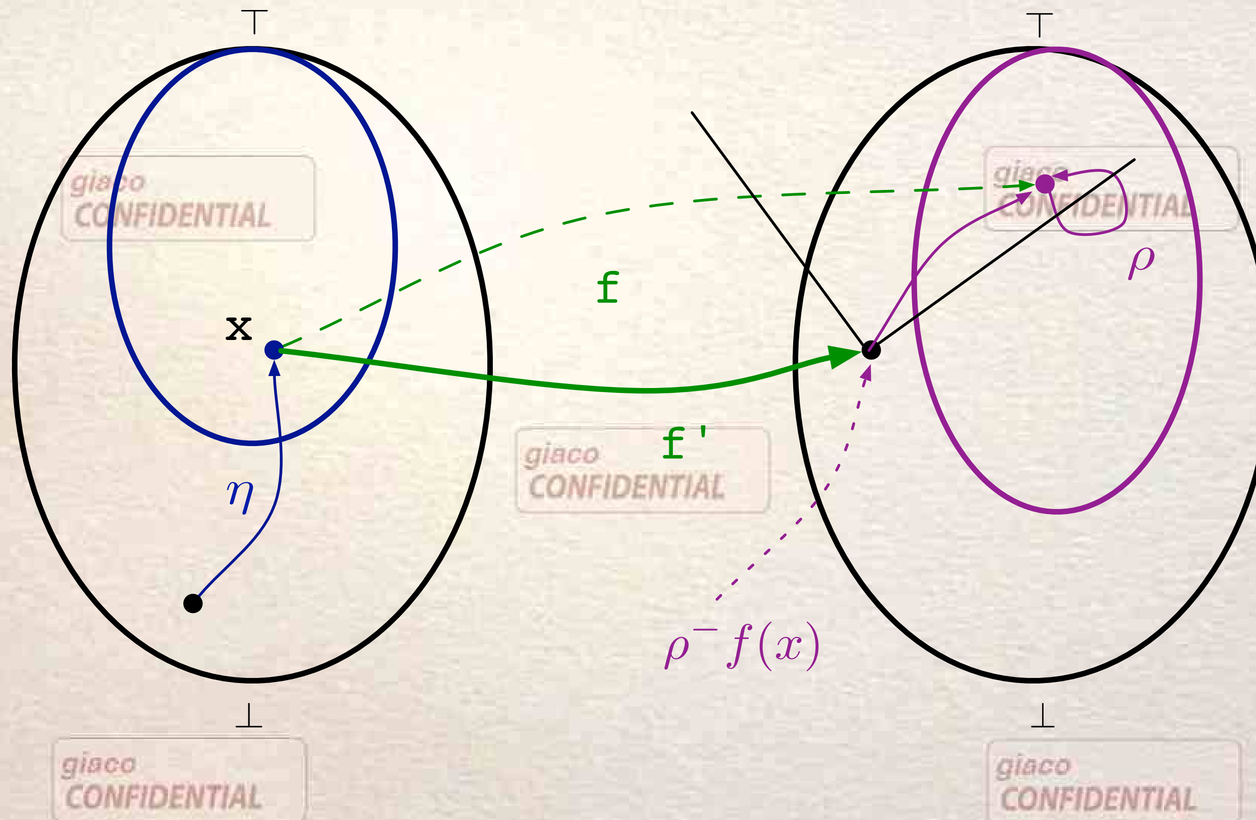


Making FORWARD IN-COMPLETENESS: Transforming the semantics upwards

$$\mathbb{O}^{\uparrow}_{\eta,\rho}(f)(x) = \begin{cases} (\rho^{+})^{+}(f(x)) = \bigvee \left\{ y \,\middle|\, \rho^{+}(y) = \rho^{+}(f(x)) \right\} & \text{if } x \in \eta \\ f(x) & \text{otherwise} \end{cases}$$

giaco CONFIDENTIAL

The Geometry of Semantics Transformers



Making FORWARD IN-COMPLETENESS: Transforming the semantics downwards

$$\mathbb{O}^{\downarrow}_{\eta,\rho}(f)(x) = \begin{cases} \rho^-(f(x)) = \bigwedge \left\{ y \,\middle|\, \rho(y) = \rho(f(x)) \right\} & \text{if } x \in \eta \\ f(x) & \text{otherwise} \end{cases}$$

giaco CONFIDENTIAL

What do we have so far?

We know how to transform semantics to make them:

- Minimally complete
- Maximally incomplete

Incompleteness is essential for obfuscation

These are not transformations but deformations!

- The semantics may change
- We have to cope with this change

giaco CONFIDENTIAL

Obfuscation as incompleteness

We transform semantics in order to induce maximal incompleteness

$$P : \left[\begin{array}{ll} & x = x * x; \\ c : & \textbf{if } 10 \leq x \leq 100 \, \{y = 5\} \, \{y = 5000\}; \\ & \textbf{return}(y) \end{array} \right.$$

$[\![P]\!]^{\iota}(x \in [5, 8]) = x \in [25, 64] \wedge y \in [5]$

$\textbf{wlp}[\![c]\!]^{\iota}(y \leq 100) = x \in [10, 100]$ and
$\textbf{wlp}[\![x = x * x]\!]^{\iota}(x \in [10, 100]) = x \in [4, 10]$.

Find c' such that

$$\begin{aligned} & \textbf{wlp}[\![c']\!]^{\iota}(x \in [10, 100]) = \\ & \quad \mathbb{O}^{\downarrow}_{\iota,\iota}(\lambda X.\, \textbf{wlp}[\![x = x * x]\!]^{\iota}(X))(x \in [10, 100]) = \\ & \quad \iota^{-}(\textbf{wlp}[\![x = x * x]\!]^{\iota}(x \in [10, 100])) = \{4, 10\} \end{aligned}$$

giaco CONFIDENTIAL

OBFUSCATION AS INCOMPLETENESS

We transform semantics in order to induce maximal incompleteness

$$P : \left[\begin{array}{ll} & x = x * x; \\ c : & \textbf{if } 10 \leq x \leq 100 \, \{y = 5\} \, \{y = 5000\}; \\ & \textbf{return}(y) \end{array} \right.$$

c' : **if** $x == 4 \vee x == 10 \, \{x = 16\} \, \{x = x * 200\}$

In order to ensure behaviour equivalence we derive

$$\begin{array}{ll} \textbf{if} & 4 \leq x \leq 10 \\ & \{x = x - (x - 4) \, \Box \, x = x - (x - 10)\} \\ & \{\textbf{nil}\} \end{array}$$

giaco CONFIDENTIAL

OBFUSCATION AS INCOMPLETENESS

We transform semantics in order to induce maximal incompleteness

$$P : \left[c : \begin{array}{l} x = x * x; \\ \textbf{if } 10 \leq x \leq 100\ \{y = 5\}\ \{y = 5000\}; \\ \textbf{return}(y) \end{array} \right.$$

The resulting obfuscated code is:

$$\tau(P) : \left[\begin{array}{l} \textbf{if} \quad 4 \leq x \leq 10 \\ \qquad \{x = x - (x - 4)\ \square\ x = x - (x - 10)\} \\ \qquad \{\textbf{nil}\}; \\ \textbf{if } x == 4 \vee x == 10\ \{x = 16\}\ \{x = x * 200\}; \\ \textbf{if } 10 \leq x \leq 100\ \{y = 5\}\ \{y = 5000\}; \\ \textbf{return}(y) \end{array} \right.$$

For $x = 7$ we have

$$[\![\tau(P)]\!]^{\iota}(x \in [5, 8]) = x \in [16, 1400] \wedge y \in [5, 5000]$$

giaco CONFIDENTIAL

Obfuscation as incompleteness

We transform semantics in order to induce maximal incompleteness

$$P : \left[c : \begin{array}{l} x = x * x; \\ \textbf{if } 10 \leq x \leq 100 \, \{y = 5\} \, \{y = 5000\}; \\ \textbf{return}(y) \end{array} \right.$$

The resulting obfuscated code is:

$$\tau(P) : \left[\begin{array}{l} \textbf{if} \quad 4 \leq [5,8] \leq 10 \\ \qquad \{x = [5,8] - ([5,8] - 4) \,\square\, x = x - (x - 10)\} \\ \qquad \{\textbf{nil}\}; \\ \{x \in [1,7]\} \\ \textbf{if } x == 4 \vee x == 10 \, \{x = 16\} \, \{x = x * 200\}; \\ \textbf{if } 10 \leq x \leq 100 \, \{y = 5\} \, \{y = 5000\}; \\ \textbf{return}(y) \end{array} \right.$$

For $x = 7$ we have

$$[\![\tau(P)]\!]^{\iota}(x \in [5,8]) = x \in [16, 1400] \wedge y \in [5, 5000]$$

giaco CONFIDENTIAL

Obfuscation as Incompleteness

We can derive a methodology for systematically making code obscure:

$P = M_1; \ldots; M_j; \{\Phi_j\}\, M_{j+1}; \ldots; M_n$

Assume the invariant Φ_j can be generated with abstract interpretation α

Find C such that:

$$\mathbf{wlp}[\![C]\!]^{\alpha}(\Phi_j) = \mathbb{O}^{\downarrow,\uparrow}_{\alpha,\alpha}(\lambda X.\, \mathbf{wlp}[\![M_j]\!]^{\iota}(X))(\Phi_j)$$

Adjust $C \rightsquigarrow C'$ in order to keep concrete observational (I/O) behaviour

$$C' \models \Phi_j$$

$\tau(P) = M_1; \ldots; C'; \Phi_j\, M_{j+1}; \ldots; M_n$

giaco CONFIDENTIAL

Thwarting Disassembly

- Confuse as much as possible the position of the instruction boundaries in a program
- Compare the set of instructions start addresses identified by a static disassembler and the "actual" instruction addresses encountered when the program is executed (maximize this difference)
- Self-reparing disassembly: even when a disassembly error occurs the disassembly ends up re-syncronizing with the actual instruction stream
- Delay the self-reparing process as much as possible

giaco CONFIDENTIAL

Thwarting Disassembly: Junk Insertion

- Introduce disassembly errors by inserting junk bytes where the disassembler expects code
  - ✔ Junk bytes are partial instructions
  - ✔ Junk bytes are never executed at run time
- Candidate block is a Basic Block that can have junk bytes inserted before (the BB preceding ends with a direct jump or a function call)
- Given a candidate block we have to determine the junk bytes to insert before it in order to delay the self-reparing process as much as possible (we insert the first k byte of a given instruction I where k maximizes the delay)

giaco CONFIDENTIAL

giaco CONFIDENTIAL

THWARTING LINEAR SWEEP

Attack limitation: Unable to distinguish data embedded in the instruction stream

Junk bytes insertion: 26% - 30% of instructions are incorrectly disassembled (candidate blocks are too far $\sim$ 30 instructions)

Increase the number of candidate blocks through branch flipping:

$bcc Addr$	$bcc L$
$L:$	$jmp Addr$
	$L:\ldots$

L is now a candidate block!!

70% of the instructions are incorrectly disassembled

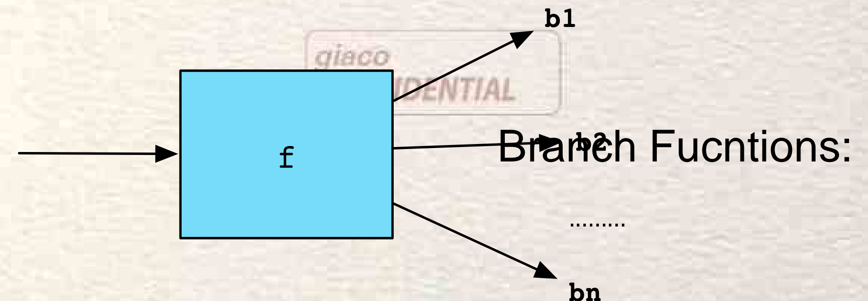
THWARTING RECURSIVE TRAVERSAL

Assumption: a function returns to the instruction following the call instruction

Branch functions:

```
a1: jmp b1    a1: call f
a2: jmp b2    a2: call f

an: jmp bn    an: call f
```



Obscure the control flow of the program

Confuse disassembly by inserting junk bytes at the points immediately after `call f`

The resulting code is more expensive (depending on the implementation of the branch functions) and therefore such transformation is not applied to “hot code” (code frequently executed)

giaco CONFIDENTIAL

Thwarting Disassembly: Experimental evaluation

- Identify the hot blocks
- Linear sweep: 75% of instructions are incorrectly disassembled
- Recursive traversal: 40% of instructions are incorrectly disassembled
- Average 52% execution speed penalty

giaco CONFIDENTIAL

DYNAMIC OBFUSCATION: DEFINITIONS

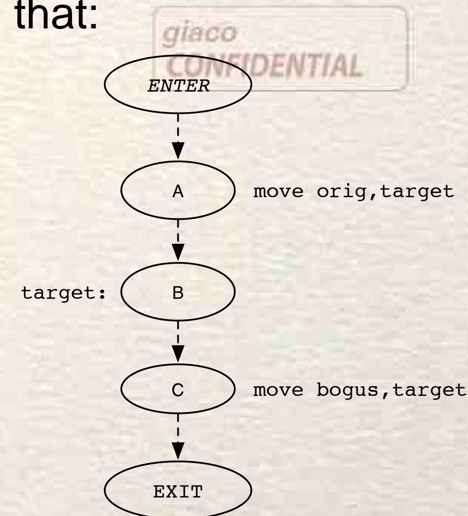
- Static obfuscations transform the code prior to execution.
- Dynamic algorithms transform the program at runtime.
- Static obfuscation counter attacks by static analysis.
- Dynamic obfuscation counter attacks by dynamic analysis
- A dynamic obfuscator runs in two phases:
  1. At compile-time transform the program to an initial configuration and add a runtime code-transformer.
  2. At runtime, intersperse the execution of the program with calls to the transformer.
- A dynamic obfuscator turns a *normal* program into a self-modifying one.

giaco CONFIDENTIAL

DYNAMIC OBFUSCATION: REPLACING INSTR.

`Obfuscate(P):`

1. Select three points A, B, and C in P, such that:
   - ✔ A strictly dominates B,
   - ✔ C strictly post-dominates B, and
   - ✔ any path from B to A passes through C.



2. Let `orig` be the instruction at B.
3. Select an instruction `bogus` of the same length as `orig`.
4. Replace `orig` at B with `bogus`.
5. At point A insert the instruction `move orig,` $v^{=B}$ where v is an opaque expression that evaluates to the address of point B.
6. Similarly, at point C insert the instruction `move bogus,` $v^{=B}$.

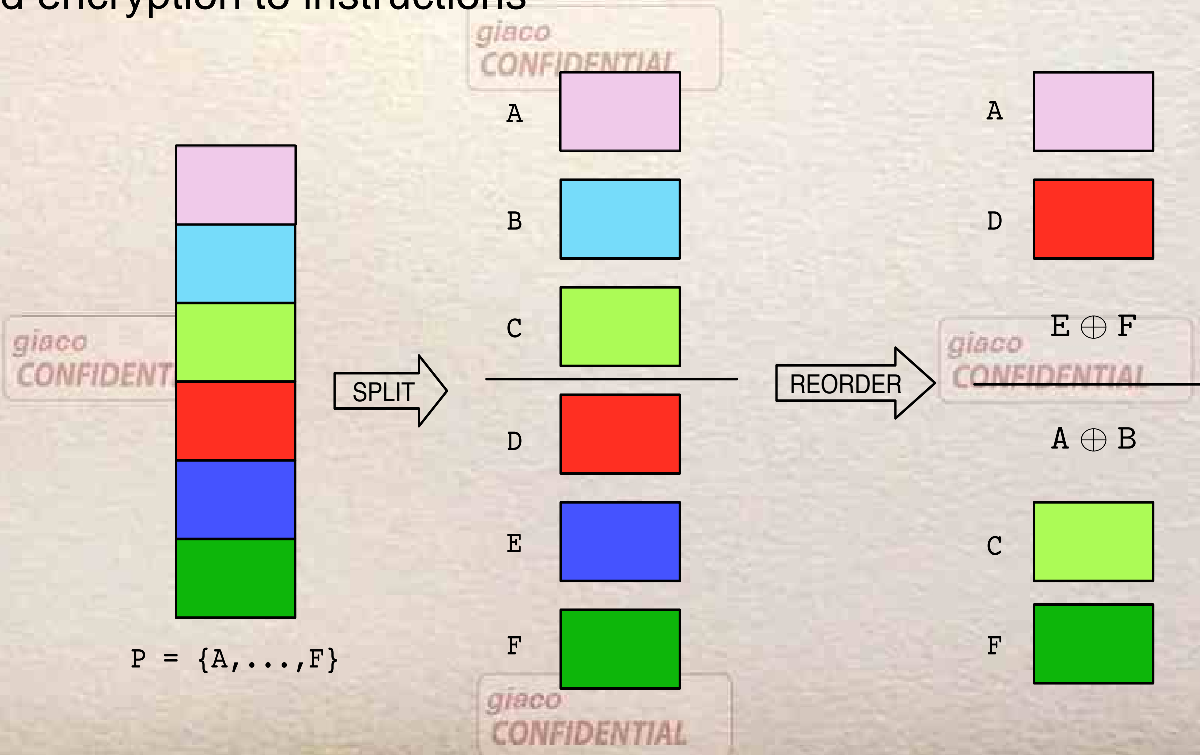
giaco CONFIDENTIAL

DYNAMIC OBFUSCATION: AUCSMITH'S ALGORITHM

D. Aucsmith, Tamper resistant SW: An implementation, Patent: 5892899, 1999

Idea:

- Split the program into pieces
- `xor` them with each other
- Add encryption to instructions



giaco CONFIDENTIAL

DYNAMIC OBFUSCATION: AUCSMITH'S ALGORITHM

D. Aucsmith, Tamper resistant SW: An implementation, Patent: 5892899, 1999

In general:

- $(A \oplus B) \oplus A = B$
- $(A \oplus B) \oplus B = A$

Idea

- Reorder blocks such that upper and lower blocks are alternated
- Execute a block and `xor` upper with lower if even iterate
- Execute a block and `xor` lower with upper if odd iterate

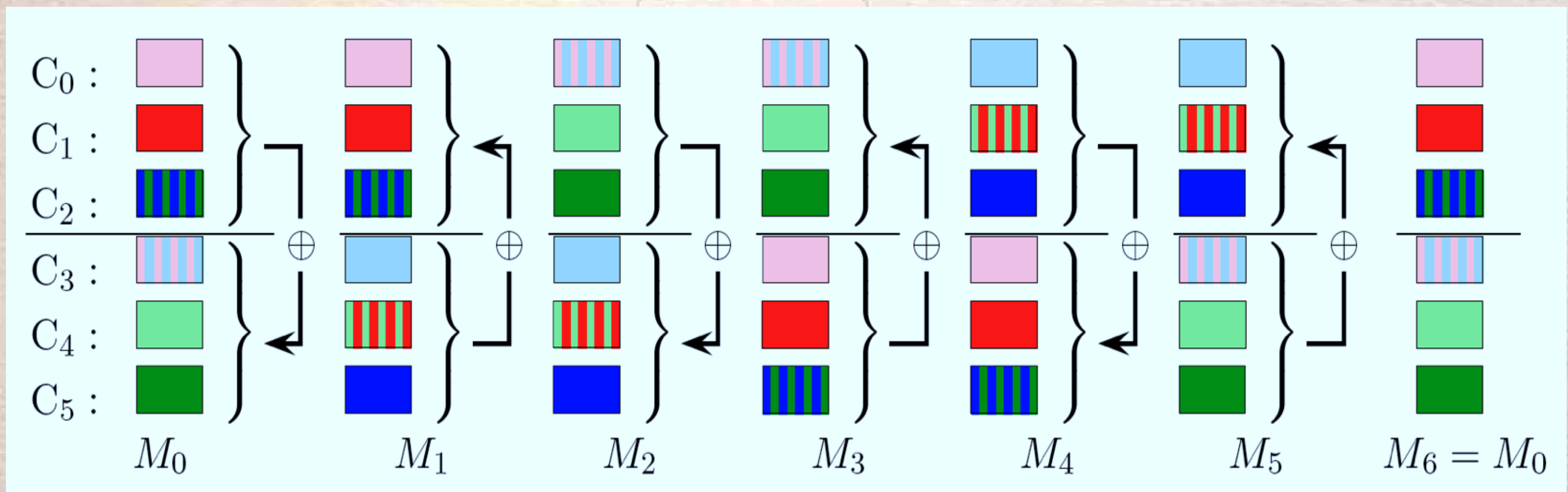
giaco CONFIDENTIAL

Dynamic obfuscation: Aucsmith's Algorithm

D. Aucsmith, Tamper resistant SW: An implementation, Patent: 5892899, 1999

In general:

- $(A \oplus B) \oplus A = B$
- $(A \oplus B) \oplus B = A$



giaco CONFIDENTIAL

giaco CONFIDENTIAL

DYNAMIC OBFUSCATION: AUCSMITH'S ALGORITHM

`Obfuscate(P):`

1. Split P into n pieces: $P_0, \ldots, P_{n-1}$.
2. Let $C_0, \ldots, C_{n-1}$ be the memory cells in which the pieces will reside at runtime. The C_i's are of equal size, large enough to fit the largest piece P_j.
3. Cells are, conceptually, divided into two spaces, upper ($C_0, \ldots, C_{n/2-1}$) and lower ($C_{n/2}, \ldots, C_{n-1}$). Each cell in the upper space partners with a cell in the lower space. Select a partner function $PF(c)$ that maps a cell number to the cell number of its partner, such as $PF(c) = c + K$, for some constant K.
4. Let $IV_0, \ldots, IV_{n-1}$ be the initial values of cells $C_0, \ldots, C_{n-1}$, respectively.
5. Initialize a set of equations $\mathcal{E}_1 = \{C_0 = IV_0, \ldots, C_{n1} = IV_{n-1}\}$ which expresses the current state of the memory cells as a function of their initial values.
6. Initialize a set of equations $\mathcal{E}_2 = \{\}$ which expresses how a piece P_i can be recovered in cleartext from the initial values $IV_0, \ldots, IV_{n-1}$.
7. Initialize a table `next` $= \langle P_0 = ?, \ldots, P_{n-1} = ?\rangle$ which maps each subprogram P_i to the cell it should jump to in order to execute P_{i+1}.
8. `make_obscure()`

DYNAMIC OBFUSCATION: AUCSMITH'S ALGORITHM

giaco CONFIDENTIAL

`make_obscure()`:

- For $p \in [0 \ldots n-1]$ do
  - Select a cell C_c to hold piece P_p in cleartext.
  - Consult $\mathcal{E}_1$ to find the current contents V of C_c. Update $\mathcal{E}_2 := \mathcal{E}_2[P-p=V]$. Using *Gaussian elimination*, try to invert $\mathcal{E}_2$ (i.e. find values for all the IV_i's). If there is no solution select another cell for P_p.
  - $\texttt{next} := \texttt{next}[P_{p-1} = C_c]$
  - For even (odd) p:s, simulate a mutation where every cell C_i in upper (lower) space is `xor`:ed with its partner cell $C_{PF(i)}$ in lower (upper) space.
  - $\mathcal{E}_1 := \mathcal{E}_1[C_{PF(i)} = C_{PF(i)} \oplus C_i]$.

DISCUSSION: THE FUCSIA IDEA

Obfuscation and Steganography by Abstract Interpretation

Define a uniform framework for information concealment in programming languages

- General enough to include most known methods
- Formal enough to provide a (possibly) provable secure environment for obfuscation and steganography
- Rich enough to provide advanced design and evaluation tools
- Practical enough to become a standard in the obfuscation and steganographic design and evaluation

The goal: develop a theory and practice for code obfuscation and steganography in order to make these technologies as practical as analogous ones in other media (e.g., in DRM of audio and video)

- The code is a new media
- Known concepts in digital media (compression, noise etc.) have to be studied on software

giaco CONFIDENTIAL

FUTURE DIRECTIONS

Move from syntactic to semantic-based metrics for potency

- measuring incompleteness
- measuring complexity of complete refinements

We need a program refinement-like calculus for code obfuscation

- A kind of Program Obfuscation by Stepwise Refinement [Wirth '71] ?
- This is an open issue....

giaco CONFIDENTIAL