IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Safe Control of Obfuscation Toolchain

Nicolas SZLIFIERSKI

nicolas.szlifierski@imt-atlantique.fr

13/12/2018

Obfuscations

Goals:

- Protect data and intellectual properties from reverse engineering (secret keys, algorithms ...)

With a reasonable cost in:

- Execution time
- Memory consumption
- Binary size

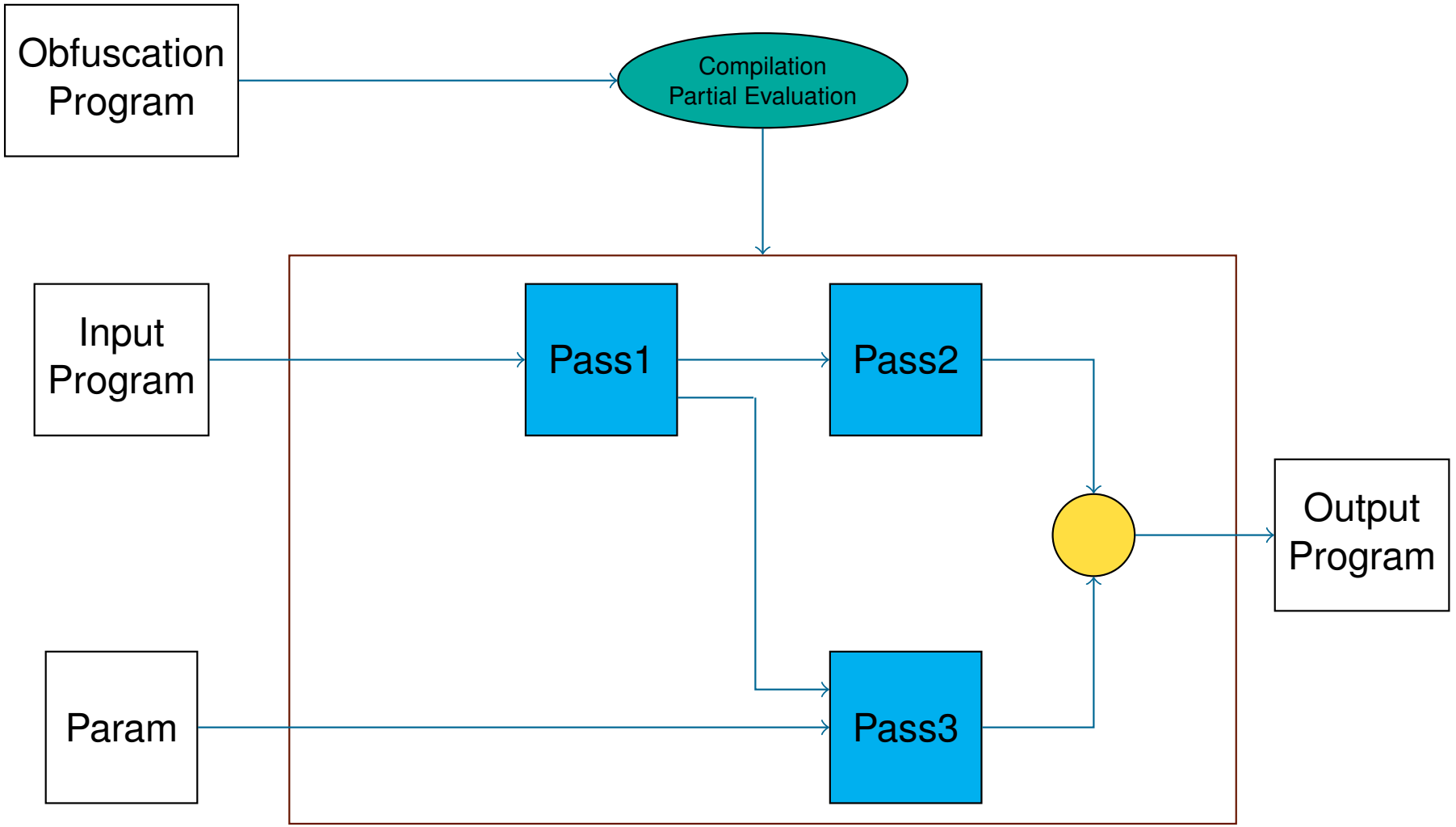
Can be done at multiple level (source, binary , IR)

- Obfuscations transformations (often) have a huge impact on performances
  - Not reasonable to apply all transformations to the whole program
  - Where and how to protect the program should be choosen wisely to attain the wanted performance/protection compromise
- Composition of transformation is not easy:
  - Order matters: Previous transformations can hinder obfuscations, and future transformations can *deobfuscate* the code
  - Transformation can generate invalid code when given inputs that don't meet certains pre-conditions
  - New code is generated during the process (and should be protected too)
  - Performance impact is hard to determine statically
- Traditionnal pass schedulers are not well fitted for obfuscations passes

Principles

- A functionnal domain-specific language for obfuscation and optimisation transformations composition
- Describes the compilation flow for each function in the program (including those generated by the process)
- Checks the correctness of the applications of those transformations (pre-conditions ...)
- Checks the obfuscation *state* of each functions (Which obfuscation have been effectively applied)
- With conditional structures and recursions to write generic compilation flow and use inputs from dynamic analysis
- Extracts information of the compilation flow (trace) for reproductability/debug purposes

Compilation flow execution



Principle

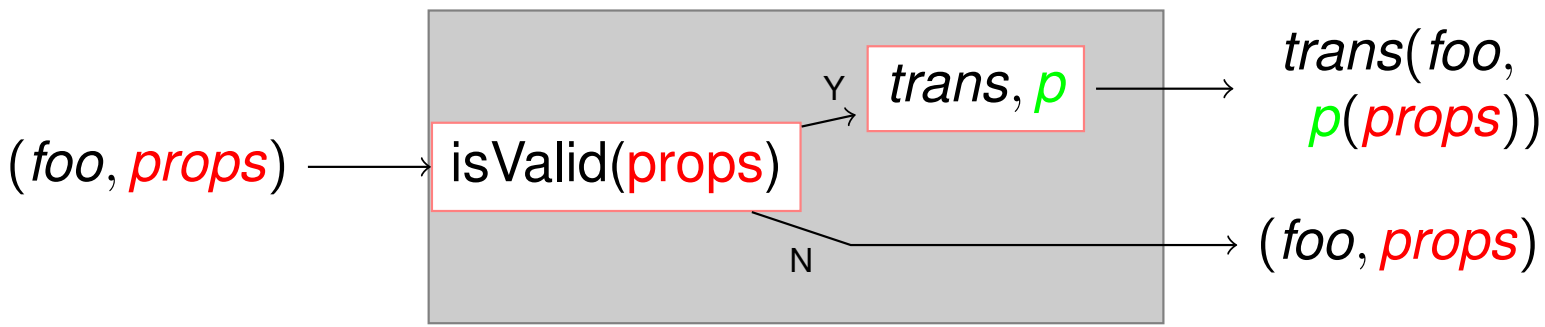
- Functions types contain **properties**
- Transformations may require functions to have certain properties to run on them (pre-conditions)
- Transformations may modify properties of functions
- Uses gradual typing to allow for both static typing (when possible) and dynamic typing

Goals

- Ensure that transformation are run and are effective
- Ensure that the final program is valid
- Extract informations on the level of obfuscation

Exemple

- Program that runs a transformation only if the function has the correct properties




```
let foo = func "foo" : props in
let result = if isValid props then trans foo else foo
```

Conclusion

- DSL for obfuscation and compilation flow
- Which ensure the correctness of the transformations flow

Current and Future work

- Determine the level of protection from the properties
- Finish implementing the execution of the control flow (in LLVM)