Epona and the Obfuscation Paradox:

Transparent for Users and Developers, a Pain for Reversers

Béatrice Creusillet, Pierrick Brunet, Adrien Guinet, Juan Manuel Martinez

quarkslab
www.quarkslab.com

Context: Code Protection

Attack Model

- Full access to binaries
- Code running on **untrusted environments**

Context: Code Protection

Attack Model

- Full access to binaries
- Code running on **untrusted environments**

Consequences

Applications can be **tampered with**, **debugged** and **reversed engineered**

- Algorithms can be reversed from binary code
- Protocols, secret keys can be extracted

The Epona Obfuscator

More than an obfuscator...

- Obfuscations, anti-debug, anti jailbreaks, ...

The Epona Obfuscator

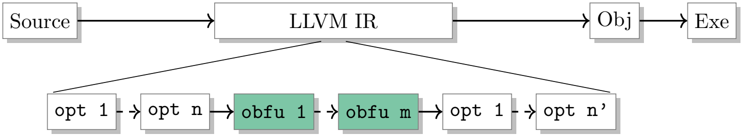
More than an obfuscator...

- Obfuscations, anti-debug, anti jailbreaks, ...
- Protection schemes tailored by the user
  - annotations
  - YAML schemes

The Epona Obfuscator

More than an obfuscator...

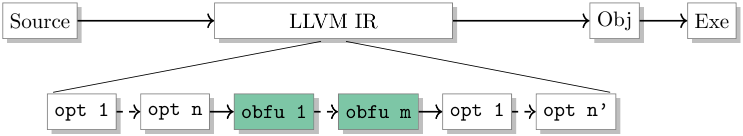
- Obfuscations, anti-debug, anti jailbreaks, ...
- Protection schemes tailored by the user
  - annotations
  - YAML schemes
- Integrated in the Clang/LLVM compiler



The Epona Obfuscator

More than an obfuscator...

- Obfuscations, anti-debug, anti jailbreaks, ...
- Protection schemes tailored by the user
  - annotations
  - YAML schemes
- Integrated in the Clang/LLVM compiler



- Reporting tools
- Verification tool

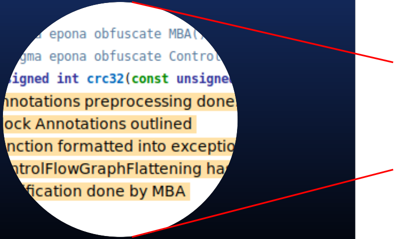
The Obfuscation Paradox

Reverser Side

User Side

The Obfuscation Paradox

Reverser Side



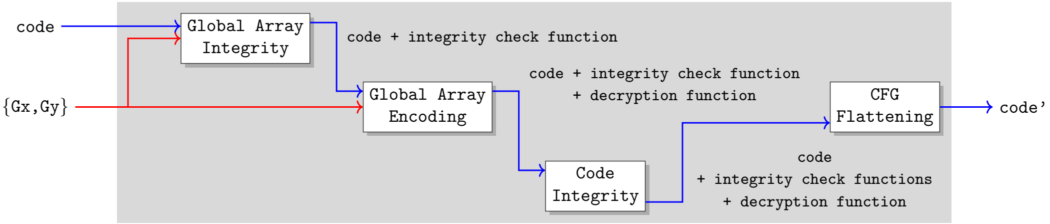
```
a epona obfuscate MBA()
gma epona obfuscate Control
signed int crc32(const unsigned
nnotations preprocessing done,
ock Annotations outlined
nction formatted into exceptio
ntrolFlowGraphFlattening ha
fication done by MBA
```

User Side

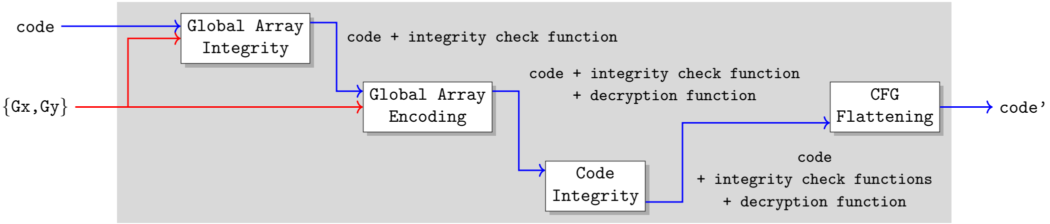
Have the protections been successfully applied?
If not, why?

Are the sensitive assets actually protected?

Using Regular LLVM Passes



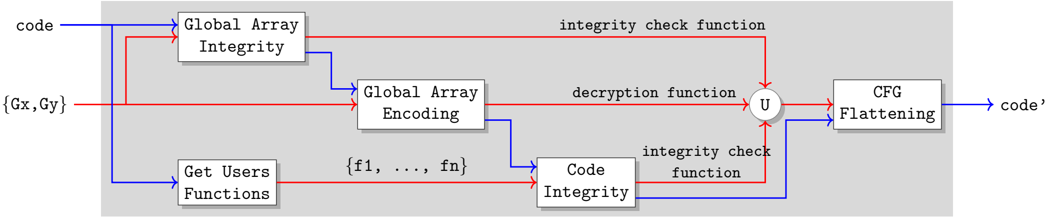
Using Regular LLVM Passes



- Can't finely target newly produced artifacts, . . . or avoid targeting them
- Communication between transformation passes through LLVM metadata only
  - not adapted to all artifacts
- Consequences on trade-off between protection and performance (and compilation time)

Fine Grain Pass Combination

- Gain access to passes artifacts (*Values*, operands,. . .)
- Improve protection/performance trade-off, and compilation time
- Ease design and reuse of protections



A Matter of Reporting

Directly to the user

- Logs (Warnings, Errors,. . .), LLVM Remarks, ...

To External Tools

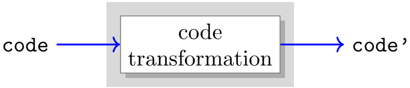
- For verification purpose

Between Passes (Protections, Optimizations)

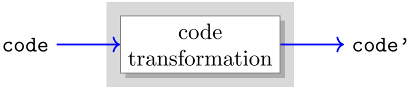
- Pass properties[1]
- Pass artifacts

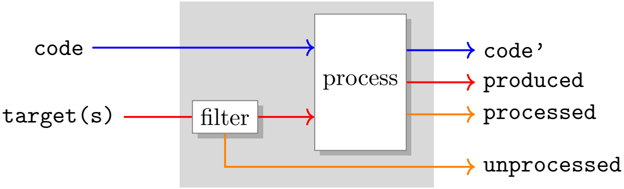
[1] *Combining Obfuscation and Optimizations in the Real World*, Scam 2018, Madrid, Spain
Serge Guelton, Adrien Guinet, Pierrick Brunet, Juan Manuel Martinez, Fabien Dagnat and Nicolas Szlifierski

Enriching the Pass Model

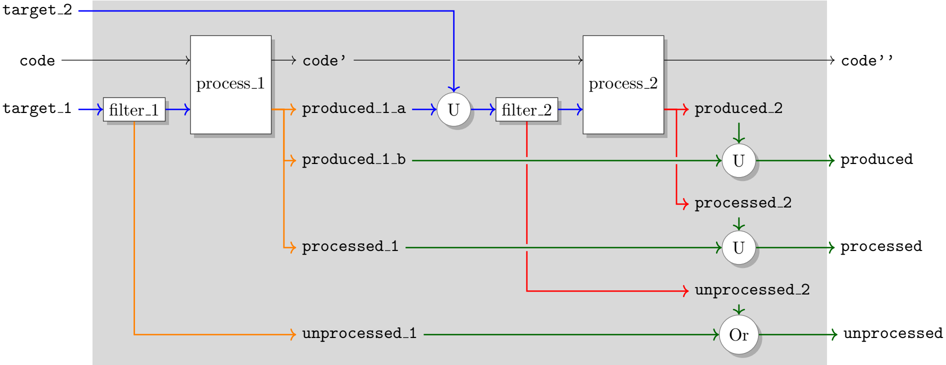


Enriching the Pass Model

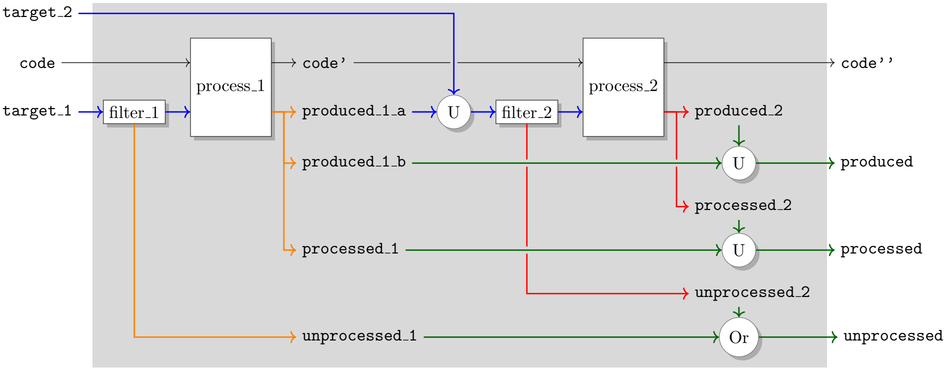




Combining Protections



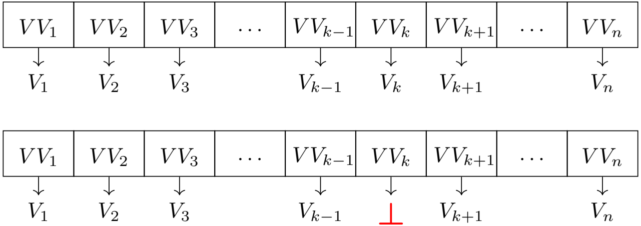
Combining Protections



- Management of reporting (produced, processed, unprocessed)
- Targets/Artifacts and reports validity
- Fitting the new pass model (filter/process): not always optimal

Artifacts Containers Validity: *Value Views*

- *Artifacts* may be destroyed by subsequent passes
- How to ensure the long term validity of artifact containers?
- At low cost...
  - Leverage LLVM call back mechanisms

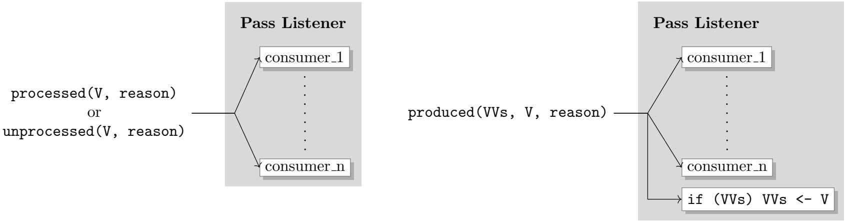


VV_1	VV_2	VV_3	$\dots$	VV_{k-1}	VV_k	VV_{k+1}	$\dots$	VV_n
$\downarrow$	$\downarrow$	$\downarrow$		$\downarrow$	$\downarrow$	$\downarrow$		$\downarrow$
V_1	V_2	V_3		V_{k-1}	V_k	V_{k+1}		V_n

VV_1	VV_2	VV_3	$\dots$	VV_{k-1}	VV_k	VV_{k+1}	$\dots$	VV_n
$\downarrow$	$\downarrow$	$\downarrow$		$\downarrow$	$\downarrow$	$\downarrow$		$\downarrow$
V_1	V_2	V_3		V_{k-1}	$\bot$	V_{k+1}		V_n

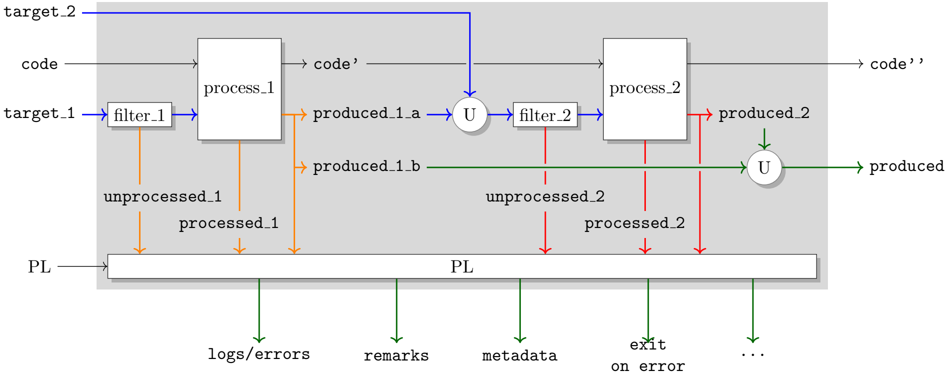
- Extensions for other kinds of artifacts (e.g. for instruction operands)

Reporting: Pass listeners and Consumers



- Reporting processed as soon as possible
  - reduce validity issues...
- PL behavior can be adapted to context
  - e.g.: silence, warn or exit on missed protection

The Whole Picture



Related Work: Pass Scheduling

- LLVM Pass Managers
  - Scope of passes not always adapted
  - No explicit management of targets
  - Metadata to convey information about parts of code
- PipsMake (Mines ParisTech)
  - Pass dependencies driven by produced resources (analyses only)
  - Scope of resources not fine enough
  - Pyps: finer grain, but external scripting language
- Adaptive compilation (e.g. Almagor *et al.*)
  - Goal: provide tailored compilation sequences
  - Search space may be enormous
  - Far from being practical yet

Related Work: Protection Pass Scheduling

- Effective Obfuscation (Heffner and Collberg)
  - Process represented by a FSA, thanks to the modelization of obfuscations (cost, potency, requirements, prohibitions, suggestions)
  - Epona's process driven by user or schemes and more dynamic (randomness).
- ASPIRE: Meta-model for software protection
  - Knowledge base for obfuscations, attack models and their links
  - Produces annotations to drive the obfuscator
  - Could Epona be a target for this kind of tool?

Status and What's Next

Over 90% of our existing passes migrated to the new architecture

Limitations

- Some pass combinations don't fit in the filter/process model
  - filtering cannot always be fully predicted before processing
  - some unprocessed reporting may come from the processing stages
  - some unwanted processed/unprocessed sequences currently issued
- Need for a kind of delayed/conditional reporting

Going further

- Provide *tunable* high-level protection schemes
- Expose fine grain passes combination to users