



Bypassing License Checking Using *DynInst*

Mihai Christodorescu
mihai@cs.wisc.edu

Computer Sciences Department
University of Wisconsin
1210 W. Dayton St.
Madison, WI 53706-1685
USA



Outline

- Background on *DynInst*
- Bypassing License Checking
 - Why? [motivation]
 - How? [approach]
 - Example [Adobe FrameMaker]
 - What? [tools / techniques]
- Future work
- Conclusion

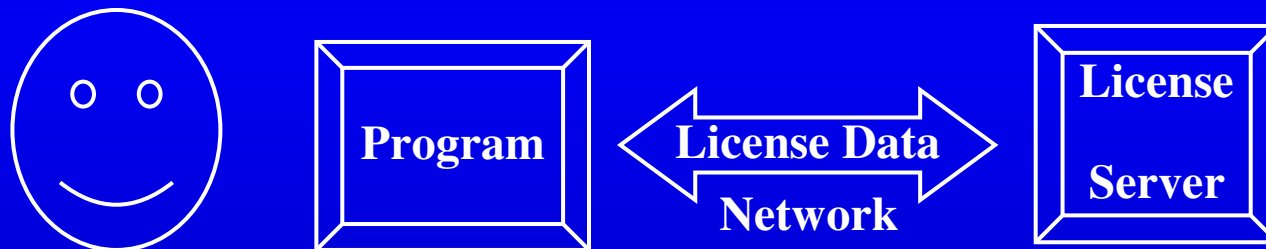
Binary Code Rewriting

- Executables are no longer black boxes
 - binaries can be modified safely at runtime
- Without access to source code
 - optimize a binary for a given input
 - change program behavior on the fly
- Open-source? What about:
 - long-running programs
 - legacy programs

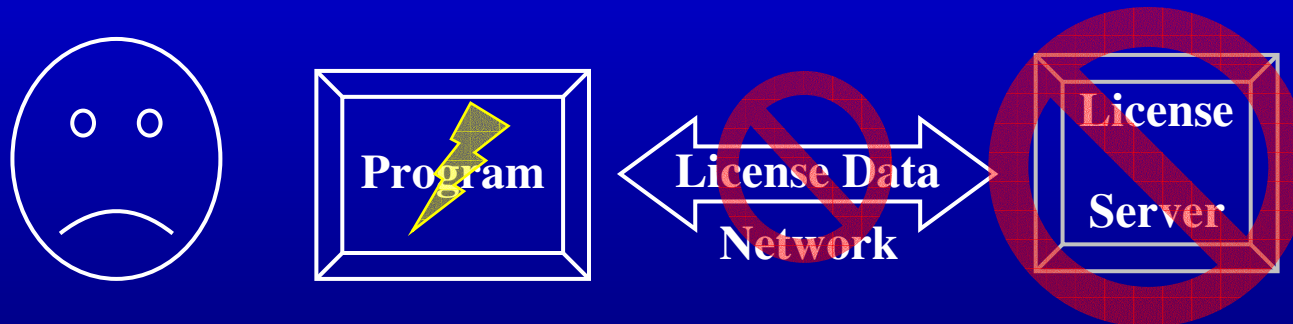
DynInst - Overview

- API to:
 - Control a process (start, stop, pause).
 - Insert code in a process.
- Current capabilities:
 - Code insertion happens at function-level (entry, exit, call site).
 - Is instruction-level granularity needed?

Why Bypass the License Check?



Normal: licensed program runs after communicates with license server.



Undesired: licensed program refuses to run if license server does not respond.

How to Bypass License Checking?

- Program = Code + Data
- Option 1: feed synthesized license data to the program
- Option 2: remove all the code that performs license checks

How...?

- Faking the license data
 - “clean” solution: capture the data once, reuse it over and over
 - problem: requires reverse engineering of client-server protocol
 - problem: license data might be time-stamped
 - problem: license data might be encrypted with a session key

How...?

- Removing the license checking code
 - removes the need of running a license server
 - problem: (very) complicated
 - problem: might alter program functionality
 - problem: not possible with current DynInst capabilities

How...?

- Middle ground solution:

Controlled Failure

- allow program to try to contact license server
- if data is OK, then nothing needs to be changed
- otherwise, force program to believe license data is OK
- limited scope of changes

Example: Adobe FrameMaker

- 2 step license verification:
 - retrieve license data from server [once]
 - check license data for correctness [often]
- allow FM to time out waiting for server
- allow FM to attempt to go into demo mode
- switch FM back to full-functionality mode
- later license checks always “succeed”

Strategies & Tools

- Complete reverse engineering:
 - not an option
 - legal problems
 - complexity (FrameMaker is a 7 MB binary!)
- Focus on certain characteristics:
 - I/O traffic
 - execution trace

I/O Monitoring

- Reduced overhead
- Low interactivity
- Can generate large amounts of data
- Cannot provide the timing information needed to modify program behavior

Function Tracing

- Fairly high overhead
- Can be interactive and incremental
(... pause trace - change - continue trace ...)
- Determining where to apply changes:
 - get trace for a successful run
 - get trace for a (forced-)failure run
 - compare to find differences
 - repeat as needed

Future Developments

- Monitor 2 processes (one successful, one failing) in parallel, and make one behave like the other one

Project Team

- Rob Iverson `riverson@cs.wisc.edu`
- Tevfik Kosar `kosart@cs.wisc.edu`
- Mihai Christodorescu `mihai@cs.wisc.edu`

Conclusions

- *DynInst* is a powerful and flexible tool
- Unlimited applicability:
 - dynamic optimizations
(measure with *Paradyn*, optimize with *DynInst*!)
 - enhance program behavior
(load new dynamic library, change calls while program is running)