SOPHOS

DSD-Tracer

Boris Lau, SophosLabs

Virus Bulletin 2007, Vienna

sophoslabs

What does DSD stand for?

- Dynamic Static DSD
  - A framework which allows data to be passed between dynamic and static analysis stages seamlessly
- To explain further…
  - Demo – GUI based on Chris Eagle's x86emu

Source of idea

- Testing discipline of software engineering
- DSD crasher
  - Christoph Csallner
  - Name of DSD-Tracer is chosen to pay tribute to this very interesting research
- Dynamic and static testing in software engineering
  - Define testing: “an examination of the characteristics of something”
  - Similar with D and S stage AV research
  - Very different requirements

How do we combine D and S?

- Different input/output
  - Dynamic analysis generally observes environmental changes
  - Static analysis look at low level binary characteristics
- In order to share information between the 2 stages…
  - One of the easiest ways is to find an intersection between domain and image of the analysis function
  - The dynamic element of DSD framework will use each state of the CPU at each tick as the basic data structure

Why do we want to do this?

- Aim to improve on traditional analysis techniques
- What is a good analysis technique?
- Coverage
  - How many of the characteristics have been explored?
  - How much has been explored?
- Accuracy
  - Is the result obscurable by the malware?
- Economy
  - How much development/human/computation time does it require?

sophoslabs

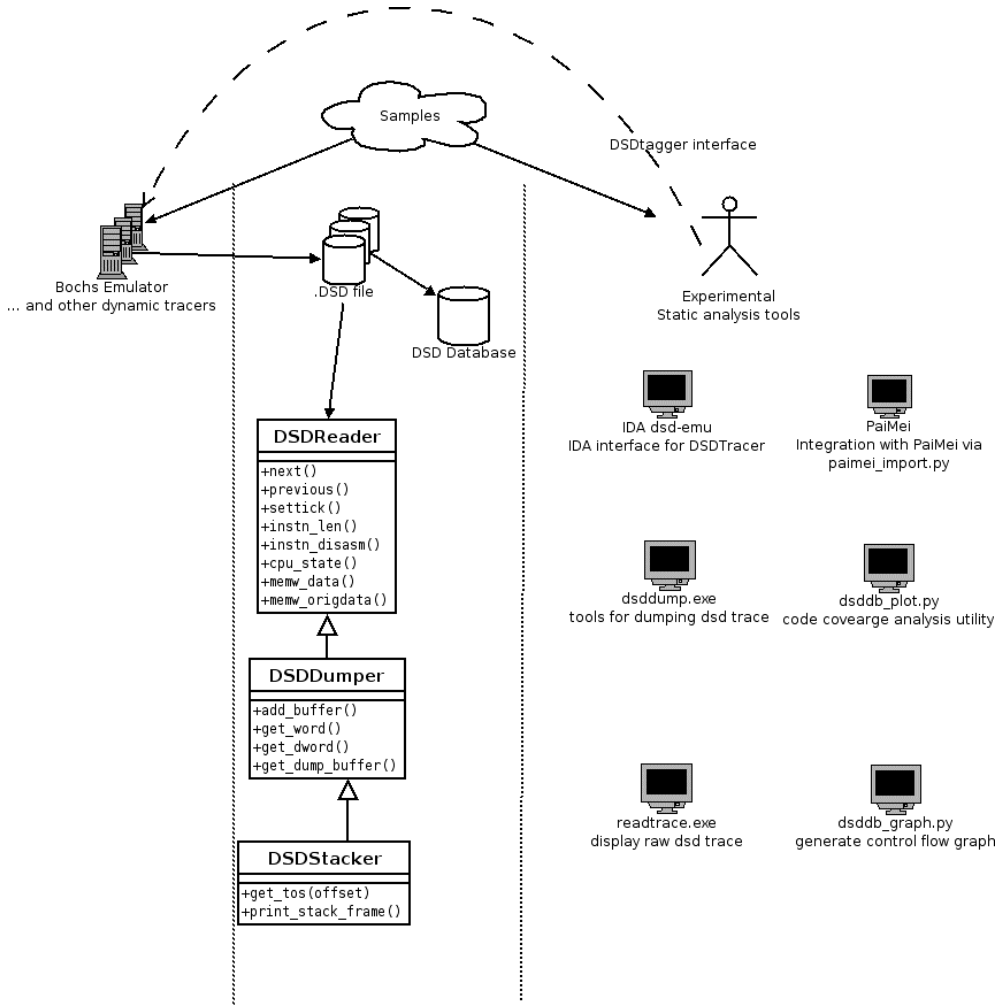
Evaluation

	Economy	Coverage	Accuracy
D (e.g. snapshot-diffing)	Good, easy to automate	No guarantee on coverage	WYSIWYG except armour techniques
S (e.g. IDA)	Difficult to automate	Good, can measure how much is covered	Difficult to armour – but no WYSIWYG
Principle of DSD	Dynamic language integration	Completeness	Cross-verify

Improve accuracy

- Record low level assembly instructions
- Compare the divergence between the 2 different dynamic analysis traces
- Assumption: low probability that an armor technique would work across different dynamic analysis techniques at the same assembly instruction

Simple architecture

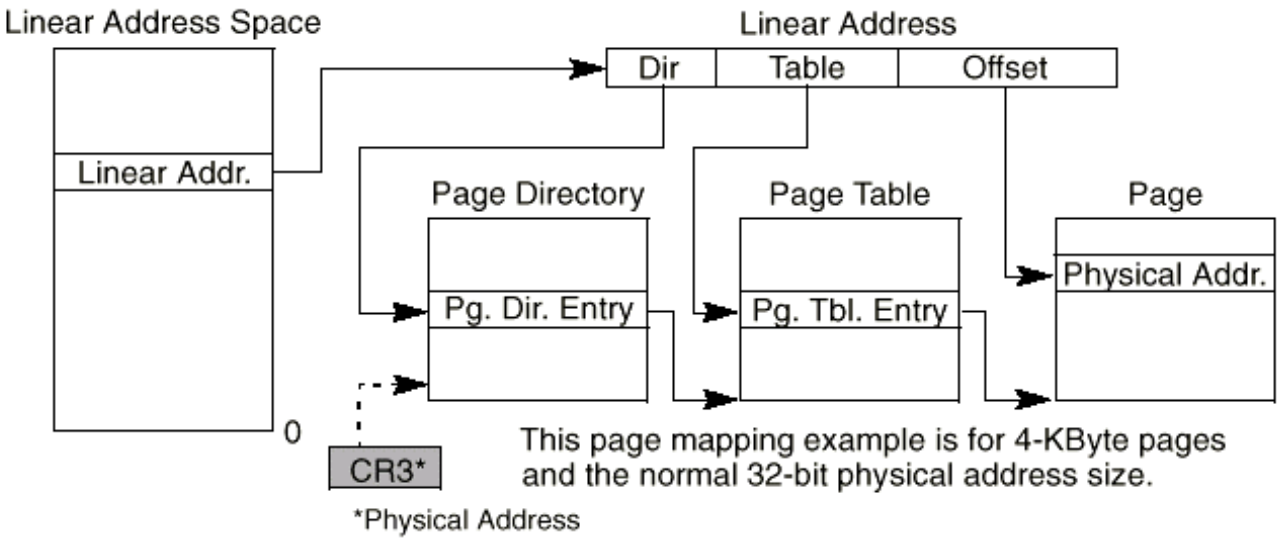


sophoslabs

Dynamic stage implementation

- Chose to instrument Virtual Machine
  - Accuracy – no OS-level, CPU-level trace detectable
  - Isolation – analysis run in a separate environment
  - Cross-platform – VM is mostly cross-platform
- Bochs – simplest to hack!
  - CPU instructions are emulated – instead of executing natively
  - Does not employ any Dynamic Binary Translation technique
    - "remarkably robust" (Tavis Ormandy , CanSecWest 2007)
  - Also contain it's own ring -1 debugger

Identify target process



- Target: Win32
- Identify the start of the process via entry point recognition
  - Careful! Need protection from static pre-analysis
- Windows separates process address space by using different Page Directory
  - Monitor CR3 which points to the current PD

sophoslabs

The DSD Trace

- Serialised list of data
  - Assembly instructions
  - CPU states at each instruction
  - Read/write to memory
  - Interrupts/Exceptions generated
- API for accessing it
  - Written in C++ (Win32/Linux compatiable)
  - Object oriented - Based class is the DSDReader class
  - Can be used by dynamic languages via swig
    - Perl, Python, Ruby, etc.

Flexibility of dynamic language

- Easy to create your own tools
  - Use your favourite language
- A brief demo
  - String information on stack
  - Very cheap to re-run the analysis by having the dynamic trace already serialised

Static analysis tools

- Tools which observe detail changes
  - IDA plugin for accessing trace
  - DSD Dumper
- Tools which abstract information
  - Control flow analysis
  - Coverage analysis
  - Data I/O analysis

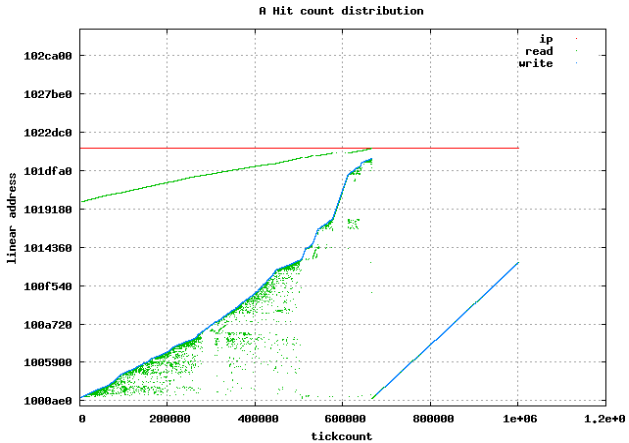
Demo

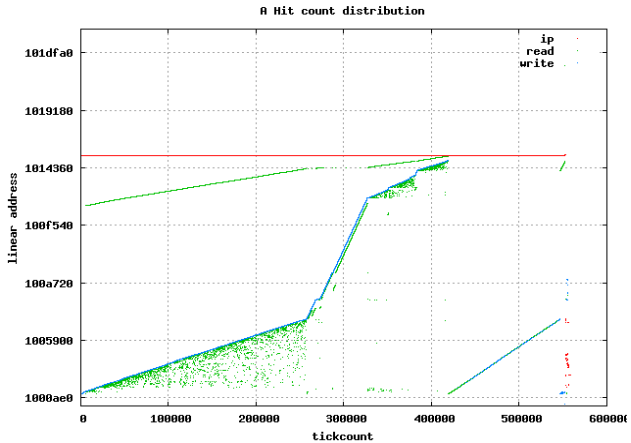
- An example of how to handle customized packer

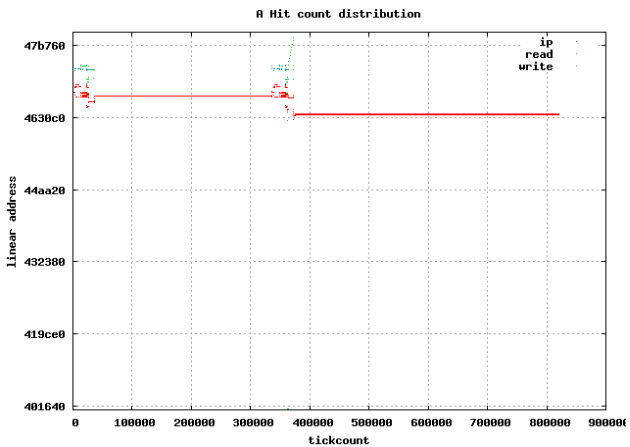
Packer identification

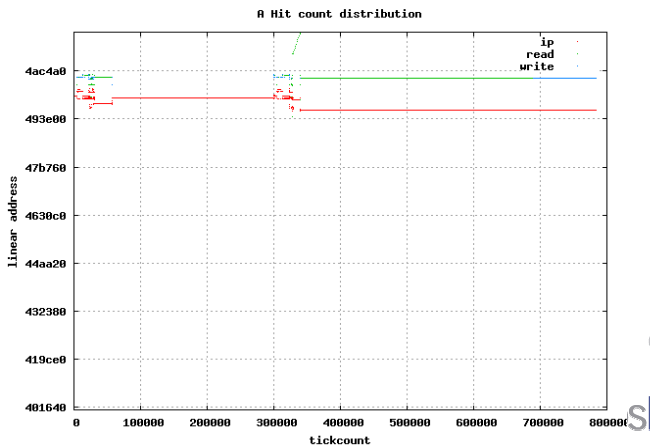
- The I/O graph could be an interesting way to differentiate packers
  - Capture information about the packer:
    - Compression ratio – gradient between read and write
    - Control flow property – how distributed is the ip
    - Section order – how the write cursor moves
    - IP vs. Relative write location
- Distinct pattern of images used by various packers

Some example graphs: UPX + Armadillo

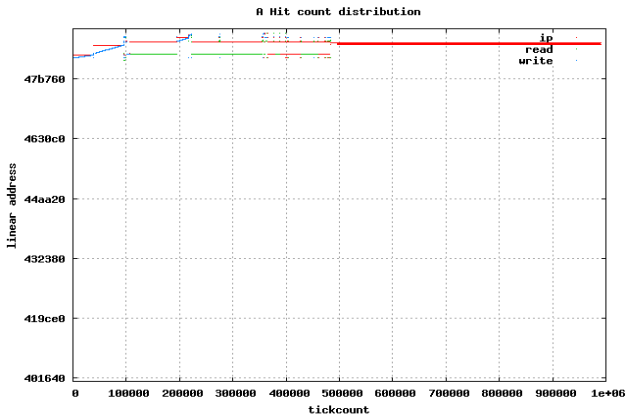


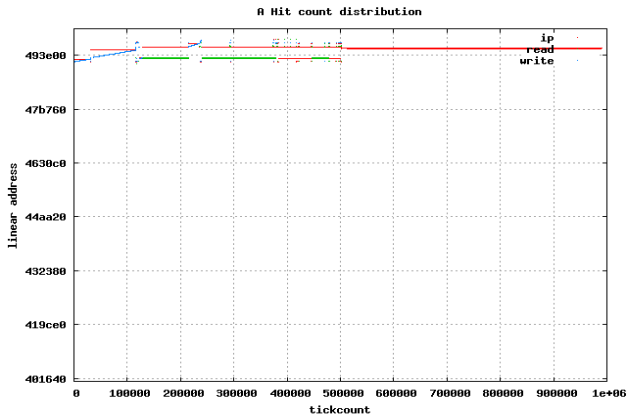


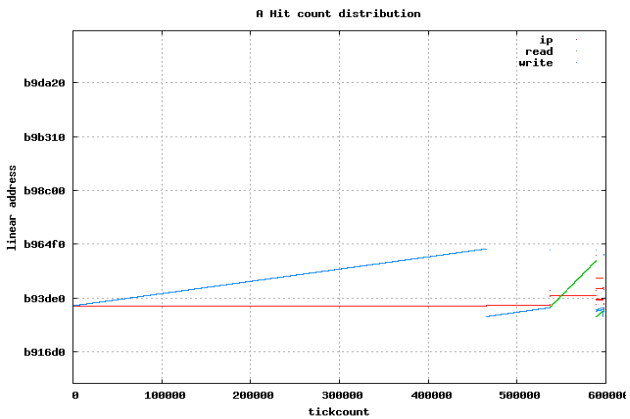


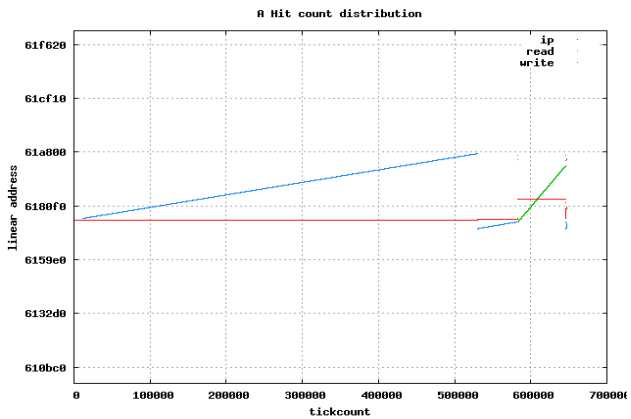


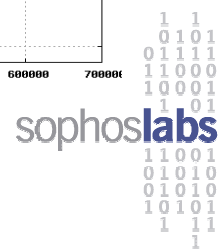
Some example graphs: Obsidium + YodaProtect



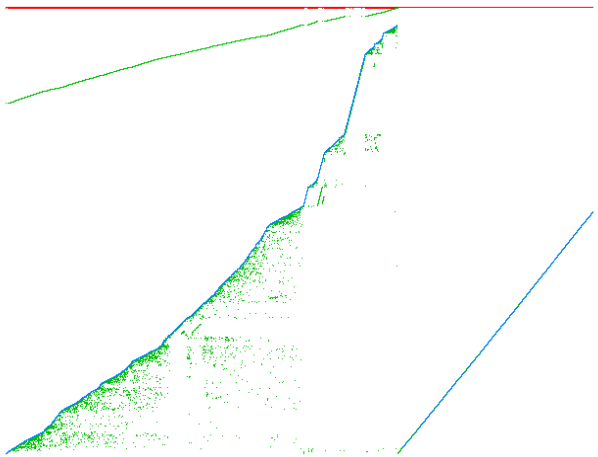


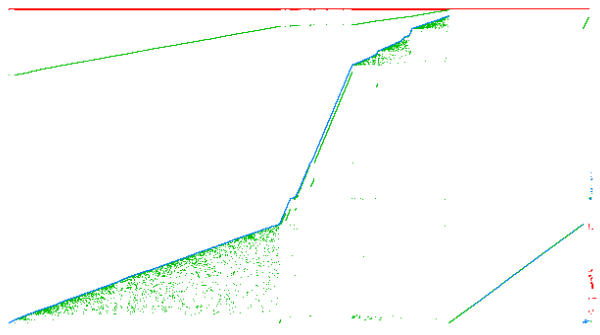






Automatic packer comparison





- Use image comparison
  - Experimented with ImageMagik's convert/compare
    - convert -blur 10x2 -fuzz 20% -trim -resize 640x480!
    - compare -metric MEPP
  - Gnuplot into cleaner state and compare

sophoslabs

Results

MEPP metrics, full image

- Obsidium.1 vs. Obsidium.2 : **1502**
  - Obsidium.1 vs. UPX: 3684
  - Obsidium.1 vs. YodaProtect: 3861
- Armadillo.1 vs. Armadillo.2 : **1965**
  - Armadillo.1 vs. UPX: 2304
  - Armadillo.1 vs. YodaProtect: 2174
  - Obsidium.1 vs. Armadillo.1: 4171

TODO

- TODO: S->D
  - Already able to change the state of the CPU via debugger
  - Add scripting capability to send keystrokes remotely to the VM
  - More research required on how to implement a suitable interface
- Better implementation of dynamic stage
  - Different techniques for generating traces
  - Handling of multiple processes and code injection

Conclusion

- Improve quality of analysis
  - Accuracy
  - Coverage
  - Economy
- Proof of concept
  - Generating trace by instrumentation of Virtual Machine
  - Provide an easy standardised API for accessing dynamic results
  - Improve ability to provide automation

Future

- Techniques for generating new analysis data
  - Backwards stepping in IDA
  - Stack based string analysis
  - Data I/O analysis
- There are certainly more possibilities….

Thank you!

- Questions?
- Email me at boris.lau(a)sophos.com