Full Potential of Dynamic Binary Translation for AV Emulation Engine

VB Conference (Oct 12, 2006)

Jim Wu
ISS X-Force
Roles of emulation in AV
Survey of software emulation technologies
Dynamic binary translation (DBT), demystified
Unique challenges and opportunities of DBT for AV
The road ahead…
Roles of emulation in AV

- Generic unpacker for unknown or modified packers.
- Detection of polymorphic malware.
- Behavioral AVs for zero-day detection.

An essential weapon in an AV arsenal
Interpretation:

- fetch-decode-execute for each instruction.
- Example: SimpleScalar®
- Based on the instruction set manual.
INSTRUCTION SET REFERENCE, A-M

AAA—ASCII Adjust After Addition

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td>AAA</td>
<td>ASCII adjust AL after addition</td>
</tr>
</tbody>
</table>

Operation

IF ((AL AND 0FH) > 9) OR (AF = 1)
THEN
    AL ← AL + 6;
    AH ← AH + 1;
    AF ← 1;
    CF ← 1;
ELSE
    AF ← 0;
    CF ← 0;
FI;
AL ← AL AND 0FH;
- **Interpretation:**
  - fetch-decode-execution for each instruction.
  - Example: SimpleScalar®
  - Based on the instruction set manual
  - Advantage: portable
  - Disadvantage: slowest (100x slower)
Dynamic Binary Translation (DBT):

- Translation in the runtime
- Execution of the generated code.
- Examples: JIT compilers, Embra.
- Translated code for “AAA”:
  - `Load_state->AAA-> save_state`
- Advantage: faster for loops
- Disadvantage: not portable
Direct execution:
- Set up a safe environment to run the sample directly.
- Example: most Ring3 code in VMware®
- New VM hardware allows the classical trap-and-emulate virtualization.
- Advantage: fastest, up to native speed
- Disadvantage: difficult to interact
Current status in AV:

- Emulation technology in AV is leaping from interpretation to DBT.
- Need to get more out of DBT
  - Example: packed samples in Wildcore
    - Currently AV engines have to use heuristics to run less instructions.
Dynamic binary translation (DBT), demystified

Dispatch Loop

Instruction in TC?

Run Blocks

Translate & Write to TC

End of Block?

Yes

No

Yes

No
Dynamic binary translation (DBT), demystified

---

1. **Dispatch Loop**
   - **Instruction in TC?**
     - Yes: **Run Blocks**
     - No: **Translate & Write to TC**

2. **End of Block?**
   - Yes
   - No: Repeat the process from the beginning (Dispatch Loop)
Dynamic binary translation (DBT), demystified

Diagram:
1. **Dispatch Loop**
2. **Instruction in TC?**
   - Yes: **Run Blocks**
   - No: **Translate & Write to TC**
3. **End of Block?**
   - Yes
   - No: Repeat from **Instruction in TC?**
Dynamic binary translation (DBT), demystified

Dispatch Loop

Instruction in TC?
  Yes
  No

Run Blocks

Translate & Write to TC

End of Block?
  Yes
  No
Dynamic binary translation (DBT), demystified

1. Dispatch Loop
2. Instruction in TC?
   - Yes: Run Blocks
   - No: Translate & Write to TC
3. End of Block?
   - Yes: Return to Dispatch Loop
   - No: Continue
Dynamic binary translation (DBT), demystified

1. Dispatch Loop
2. Instruction in TC?
   - Yes → Run Blocks
   - No → Translate & Write to TC
3. End of Block?
   - Yes
   - No → Return to Dispatch Loop
Dynamic binary translation (DBT), demystified

- **Dispatch Loop**
  - **Instruction in TC?**
    - Yes: Run Blocks
    - No: Translate & Write to TC
  - End of Block?
    - Yes: Finish
    - No: Go back to Instruction in TC?
Dynamic binary translation (DBT), demystified

Dispatch Loop

Instruction in TC?

Run Blocks

Translate & Write to TC

Yes

End of Block?

No

Yes

No
Dynamic binary translation (DBT), demystified

Dispatch Loop

Instruction in TC?

Run Blocks

Translate & Write to TC

End of Block?

Yes

No

Yes

No
Dynamic binary translation (DBT), demystified

Dispatch Loop

Instruction in TC?

Yes

Run Blocks

No

Translate & Write to TC

End of Block?

Yes

No
Translation Unit:

- Simple/repeatable for most instructions
- Example: “add edx, ecx”

```
TRANS_M2R(MOV, EBX, &(regs->ECX));
TRANS_R2M(ADD, &(regs->EDX), EBX);
```

⇒ write “01 1D xxxxxxxx” to TC

MERGE_EFLAGS();
Typical memory design:
- \((0 + 0 + 30)/3 = 10\)

Four read/write checks:
- Segment limit, HW breakpoint, split page access?
- Real memory allocated?

Two write-specific checks:
- Writable? Self-modifying-code?
- **Improved memory design -- method #1**
  - Skip segment limit check if the segment is flat.
  - Hardware breakpoint becomes page based.
Improved memory design – method #2

- Allocate reserved memory to avoid split page access.
- Use real exception if memory isn’t committed
Improved memory design – method #3

- Use hardware protection instead of insertion of explicit address checks.
- Require Ring0 programming.
- Provide the best of DBT and direct execution.

"Improved memory design – method #3

- Use hardware protection instead of insertion of explicit address checks.
- Require Ring0 programming.
- Provide the best of DBT and direct execution.

"
- **Self-modifying code**
  - Write to translated code
  - Entry in TC becomes stale:
    - Flash the whole TC
    - Flash the block
    - Complicate SMC check
    - Flash the page
    - Simplify SMC check

```plaintext
Exceed Segment Limit?  Yes  Access Violation
\[2\]
Hit a Breakpoint?  Yes  Breakpoint Exception
\[2\]
Split Page Access?  Yes  Split Page Access
\[2\]
Real Memory Allocated?  No  Allocate Real Memory for this Virtual Page
\[2\]
Page Writable?  No  Access Violation
\[2\]
Self-Modifying?  Yes  Flush Translated Block or Page
\[2\]
Access Memory
```
Unique challenges and opportunities of DBT for AV

- Translation time can’t be the bottleneck
  - Some optimizations might not worth the effort.
  - Translation might cause SMC on host machine.

- Build DBT on top of the interpreter:
  - Allow quick proof-of-concept of DBT.
    - Pack some large executables with UPX.
    - Just translate the UPX unpacking code in POC.
  - Only translate frequent loops.
  - Knowledge of the current state during translation.
The road ahead…

- Continue the transition from interpretation to DBT.
- Continue to squeeze more out of DBT.
- Explore the impact of hardware virtualization for emulation in AV.
- More research and collaboration in emulation.
Questions?

- Thank you!
- Jim Wu, ISS X-Force
  - jwu@iss.net or jywuc@yahoom.com