Dynamic Binary Instrumentation for Deobfuscation and Unpacking

Jean-Yves Marion, Daniel Reynaud

Nancy University - Loria
{jean-yves.marion|reynaud}@loria.fr
http://lhs.loria.fr
http://indefinitestudies.org

November 18, 2009
Outline

1. What is DBI
   - Tools and related work
   - What is DBI

2. How it works
   - Full control without emulation
   - How to achieve full control
   -Pros and cons

3. Demo: Javascript Deobfuscation
   - Building a Javascript DBI engine
   - Adding deobfuscation
   - Problems

4. Demo: x86 unpacking (and stuff)
Tools and related work

The most famous (general purpose) DBI tools are currently:

- Pin (by Intel, proprietary)
- Valgrind (FOSS)
- DynamoRIO (by VMware, FOSS)

Some reverse engineering projects are now using DBI:

- for vulnerability research (Microsoft’s SAGE, Sogeti’s Fuzzgrind)
- for exploit development (Sean Heelan)
- for unpacking (Piotr Bania, Danny Quist)
Definition

(informal) Dynamic Binary Instrumentation is

- a program transformation
- that gives you full control over the execution of a program
- with no need for architectural support

Note that:

- virtualization *does not provide full control*
- emulation (== interpretation) is *not a program transformation*
- debugging *requires architectural support*
- dynamic binary translation matches this definition, so QEMU can be seen as a whole-system DBI engine
Overview

Debugging

Emulation

Soft interpreter or virtualization

DBI
Outline

1. What is DBI
   - Tools and related work
   - What is DBI

2. How it works
   - Full control without emulation
   - How to achieve full control
   - Pros and cons

3. Demo: Javascript Deobfuscation
   - Building a Javascript DBI engine
   - Adding deobfuscation
   - Problems

4. Demo: x86 unpacking (and stuff)
DBI achieves the same level of control as an emulator (== interpreter), but

- DBI relies on a pre-existing, unmodified interpreter
- so there is no need to know / support the full semantics of the architecture (for x86, this is a massive benefit)
- and no requirement for debugging support (could be interesting for GPUs)
How to achieve full control

Define a static analysis $f(code)$ that returns $instrumented\_code$:

1. parse code
2. (modify user-defined program points)
3. if new\_code can potentially be executed, replace it with $f(new\_code)$
4. run $f(code)$

This can be seen as JIT compilation...
... or as virus-like program infection (hijack the entry point and retain control)
Pros and cons

Pros:
- it is a program transformation
- performance

Cons:
- it is a program transformation
- performance

...huh?
Outline

1. What is DBI
   - Tools and related work
   - What is DBI

2. How it works
   - Full control without emulation
   - How to achieve full control
   - Pros and cons

3. Demo: Javascript Deobfuscation
   - Building a Javascript DBI engine
   - Adding deobfuscation
   - Problems

4. Demo: x86 unpacking (and stuff)
Building a Javascript DBI engine

```javascript
instrument = function (script) {
  var result = script;
  result = result.replace(
    /eval\((/g,
    "instrument("
  );

  // additional result.replace() will
  // instrument the initial script

  return eval(result);
}
```

Jean-Yves Marion, Daniel Reynaud
Dynamic Binary Instrumentation for Deobfuscation and Unpacking
Adding deobfuscation

Malicious scripts often use `document.write()` to add other scripts and invisible iframes in hacked pages. Let’s instrument that:

```javascript
result = result.replace(/document.write\((/g, "log(");
```

(note to self: do the demo)
Other interesting program points in malicious Javascript:

- changes to `location.href`
- calls to `new Image()`
- calls to `new ActiveXObject()`
- calls to `setTimeout()` and `setInterval()` (sometimes used instead of `eval()`)
- (add your favorite tricks here)

(note to self: do the other demo)
Problems

To make this operational, we need to:

- replace tokens, not regexps in strings
- prevent other ways to execute dynamic code (new Function(), top/window/self/parent.eval()...)  
- manage code introspection
- make the instrumenter self-instrumentable
Outline

1. What is DBI
   - Tools and related work
   - What is DBI

2. How it works
   - Full control without emulation
   - How to achieve full control
   - Pros and cons

3. Demo: Javascript Deobfuscation
   - Building a Javascript DBI engine
   - Adding deobfuscation
   - Problems

4. Demo: x86 unpacking (and stuff)
x86 unpacking

What happens when you extract a run trace with DBI and import it in a disassembler?
Malware analysis

Pin has not been made for malware analysis, but...

- it can successfully instrument many packers (14/16 tested): AcProtect, Aspack, Expressor, FSG, Mew, Molebox, Npack, Packman, Pec2, Pelock, RLPack, UPX, Winupack, Yoda Protector...
- it works on approximately 80% of malware samples (tested on 100k samples)

Full results posted at http://indefinitestudies.org
Conclusion

Instrumentation is a **high-level, elegant** approach to solve low-level problems.

- it is a mix of **static** and **dynamic** analysis
- it will work on **any language / architecture**
- it is **algorithmically sound** (unlike event-based approaches such as debugging)
- **Outrageous claim**: it is the **fastest** method for fine-grained analysis
Conclusion Bis

I GPL’d the tools for the conference:

- Crème Brûlée (Javascript DBI):
  http://code.google.com/p/cremebrulee/

- TarteTatinTools (TraceSurfer and other pintools):
  http://code.google.com/p/tartetatintools/