Juan Antonio Artal‡, Ricardo J. Rodríguez†, José Merseguer‡

© All wrongs reversed
jaartal@gmail.com, rjrodriguez@fi.upm.es, jmerse@unizar.es
©RicardoJRdez ※ www.ricardojrodriguez.es

† Universidad Politécnica de Madrid
Madrid, Spain

‡ Universidad de Zaragoza
Zaragoza, Spain

June 21th, 2013

3rd Edition of Hack in Paris
Sequoia Lodge Hotel, Disneyland Paris
CLS member since early beginnings (2000)
Ph.D. student at University of Zaragoza
Working currently for Technical University of Madrid
  - Performance analysis of complex systems
  - Secure software engineering
  - Fault-Tolerant systems (design and analysis)
  - Malware analysis (techniques and relative stuff)
  - Safety analysis in component-based systems
CLS member since early beginnings (2000)

Ph.D. student at University of Zaragoza

Working currently for Technical University of Madrid

- Performance analysis of complex systems
- Secure software engineering
- Fault-Tolerant systems (design and analysis)
- Malware analysis (techniques and relative stuff)
- Safety analysis in component-based systems

My Ph.D. viva is next Monday! Cross fingers!! 🎉
Development Code License

- **GPL v3**
  (http://gplv3.fsf.org/)

- **Intel Open Source License**
  (http://opensource.org/licenses/intel-open-source-license.html)

- Specified in each source file

Source available at

http://webdiis.unizar.es/~ricardo/files/HIP2013.tar.gz
(VS2008 project + this slides)
Development Code License

- GPL v3
  (http://gplv3.fsf.org/)
- Intel Open Source License
  (http://opensource.org/licenses/intel-open-source-license.html)
- Specified in each source file

Source available at

http://webdiis.unizar.es/~ricardo/files/HIP2013.tar.gz
(VS2008 project + this slides)
no add-ons...trust me 😊
Outline

1. An Introduction to DBI
   - What (the hell) is Dynamic Binary Instrumentation (DBI)?
   - How does DBI work?
   - Uses of DBI in Computer Security

2. DBI Frameworks
   - DBI Framework: What is?
   - Types of DBI frameworks
   - Analysis and Comparative

3. Applying DBI to Computer Security...
   - Developing DBAs with Pin: Pintools
   - DBI vulnerability search
   - Taint analysis
   - Reverse Engineering

4. Conclusions and Acknowledgments
Outline

1. An Introduction to DBI
   - What (the hell) is Dynamic Binary Instrumentation (DBI)?
   - How does DBI work?
   - Uses of DBI in Computer Security

2. DBI Frameworks
   - DBI Framework: What is?
   - Types of DBI frameworks
   - Analysis and Comparative

3. Applying DBI to Computer Security...
   - Developing DBAs with Pin: Pintools
   - DBI vulnerability search
   - Taint analysis
   - Reverse Engineering

4. Conclusions and Acknowledgments
DBI: Dynamic Binary Instrumentation

Main Words

Instrumentation
Dynamic
Binary
DBI: What is? (I)

DBI: Dynamic Binary Instrumentation

Main Words

- Instrumentation ??
- Dynamic ??
- Binary ??
DBI: What is? (II)

Instrumentation?

- **“Being able to observe, monitor and modify the behaviour of a computer program”** (Gal Diskin)
- **Arbitrary addition of code** in executables to collect some information
DBI: What is? (II)

Instrumentation

- “Being able to **observe, monitor and modify the behaviour** of a computer program” (Gal Diskin)
- **Arbitrary addition of code** in executables to collect some information
- Analyse and control **everything around an executable code**
  - Collect some information
  - Arbitrary code insertion
DBI: What is? (III)

Instrumentation ??
Dynamic ??
Binary ??
DBI: What is? (III)

Instrumentation  What is happening...
Dynamic        ??
Binary          ??
DBI: What is? (III)

**Instrumentation**  What is happening...??
**Dynamic**  ??
**Binary**  ??
DBI: What is? (IV)

Dynamic?

Code analysis

- **Static**
  - BEFORE execution
  - All possible execution paths are explored → not extremely good for performance

- **Dynamic**
  - DURING the execution
  - Just one execution path (it may depend on the input data!)
DBI: What is? (V)

**Instrumentation**

What is happening... 

**Dynamic**

??

**Binary**

??
An Introduction to DBI

What (the hell) is Dynamic Binary Instrumentation (DBI)?

DBI: What is? (V)

Instrumentation
Dynamic
Binary

What is happening... DURING the execution... ??
DBI: What is? (V)

Instrumentation   What is happening... 
Dynamic           DURING the execution... 
Binary             ??
DBI: What is? (IV)

Binary?

Dynamic analysis

- **Source code available**
  - Source code
  - Compiler
- **No source code** (common case 😊)
  - Binary
    - Static (i.e., creating a new binary – with extras)
    - Dynamic
  - Environment
    - Emulation
    - Virtual
  - Debugging
**DBI: What is? (VI)**

**Instrumentation**

**Dynamic**

**Binary**

What is happening... DURING the execution... ??
Instrumentation Dynamic Binary

What is happening... DURING the execution... of a binary (executable)...
### Binary instrumentation: advantages

- Programming language (totally) **independent**
- **Machine-mode** vision
- We can instrument **proprietary software**
### Binary instrumentation: advantages

- Programming language (totally) independent
- Machine-mode vision
- We can instrument proprietary software

### Dynamic Instrumentation: advantages

- No need to recompile/relink each time
- Allow to find *on-the-fly code*
- Dynamically generated code
- Allow to instrument a process in execution already (*attach*)
DBI: What is? (IIIX)

DBI disadvantages

Main disadvantages

- **Overhead** (by the instrumentation during execution)
- **↓ performance** (analyst hopelessness!)
Recall: arbitrary code addition during the execution of a binary
How does DBI work? (I)

- Recall: arbitrary code addition during the execution of a binary
- What do I insert? → instrumentation function

Running code
How does DBI work? (I)

- Recall: arbitrary code addition during the execution of a binary
- What do I insert? → instrumentation function
- Where? → addition places
How does DBI work? (II)
Placing DBI in the context of dynamic analysis

Definition (informal)
- Executable transformation
- Total control over execution
- No need of architectural support

Virtualization
- Total control?

Emulation
- Executable transformation

Debugging
- Architectural support (a must...)

J-Y. Marion, D. Reynaud
Dynamic Binary Instrumentation for Deobfuscation and Unpacking
DeepSec, 2009

J.A. Artal, R.J. Rodríguez, J. Merseguer
DBI for Computer Security: Uses and Comparative
June 21th, 2013 16 / 44
Uses of DBI in Computer Security (I)
Non security-related uses

- Code coverage and metrics
Uses of DBI in Computer Security (I)

Non security-related uses

- Code coverage and metrics
- Call-graphs generation
Uses of DBI in Computer Security (I)

Non security-related uses

- Code coverage and metrics
- Call-graphs generation
- Memory leaks detection
Uses of DBI in Computer Security (I)
Non security-related uses

- Code coverage and metrics
- Call-graphs generation
- Memory leaks detection
- Instruction profiling
Uses of DBI in Computer Security (I)

Non security-related uses

- Code coverage and metrics
- Call-graphs generation
- Memory leaks detection
- Instruction profiling
- Data dependency profiling
Uses of DBI in Computer Security (I)
Non security-related uses

- Code coverage and metrics
- Call-graphs generation
- Memory leaks detection
- Instruction profiling
- Data dependency profiling
- Threads profiling
Uses of DBI in Computer Security (I)
Non security-related uses

- Code coverage and metrics
- Call-graphs generation
- Memory leaks detection
- Instruction profiling
- Data dependency profiling
- Threads profiling
- Race conditions detection
Uses of DBI in Computer Security (I)

Non security-related uses

- Code coverage and metrics
- Call-graphs generation
- Memory leaks detection
- Instruction profiling
- Data dependency profiling
- Threads profiling
- Race conditions detection

**Computer Architecture:**
Uses of DBI in Computer Security (I)

Non security-related uses

- Code coverage and metrics
- Call-graphs generation
- Memory leaks detection
- Instruction profiling
- Data dependency profiling
- Threads profiling
- Race conditions detection

Computer Architecture:

- Trace generators (memory)
Uses of DBI in Computer Security (I)

Non security-related uses

- Code coverage and metrics
- Call-graphs generation
- Memory leaks detection
- Instruction profiling
- Data dependency profiling
- Threads profiling
- Race conditions detection
- Computer Architecture:
  - Trace generators (memory)
  - Branch (and cache) predictors
Uses of DBI in Computer Security (I)

Non security-related uses

- Code coverage and metrics
- Call-graphs generation
- Memory leaks detection
- Instruction profiling
- Data dependency profiling
- Threads profiling
- Race conditions detection

Computer Architecture:
- Trace generators (memory)
- Branch (and cache) predictors
- Memory failures recovery
Uses of DBI in Computer Security (I)

Non security-related uses

- Code coverage and metrics
- Call-graphs generation
- Memory leaks detection
- Instruction profiling
- Data dependency profiling
- Threads profiling
- Race conditions detection

Computer Architecture:
- Trace generators (memory)
- Branch (and cache) predictors
- Memory failures recovery
- Simulation of speculation strategies
Uses of DBI in Computer Security (I)

Non security-related uses

- Code coverage and metrics
- Call-graphs generation
- Memory leaks detection
- Instruction profiling
- Data dependency profiling
- Threads profiling
- Race conditions detection
- Computer Architecture:
  - Trace generators (memory)
  - Branch (and cache) predictors
  - Memory failures recovery
  - Simulation of speculation strategies
Uses of DBI in Computer Security (II)

Security-related uses

- Data control flow analysis
Uses of DBI in Computer Security (II)

Security-related uses

- Data control flow analysis
- Vulnerability detection
Uses of DBI in Computer Security (II)

Security-related uses

- Data control flow analysis
- Vulnerability detection
- Test cases / fuzzing generation
Uses of DBI in Computer Security (II)

Security-related uses

- Data control flow analysis
- Vulnerability detection
- Test cases / fuzzing generation
- Advance monitoring (NSA way)
Uses of DBI in Computer Security (II)

Security-related uses

- Data control flow analysis
- Vulnerability detection
- Test cases / fuzzing generation
- Advance monitoring (NSA way)
- Reverse Engineering
Uses of DBI in Computer Security (II)

Security-related uses

- Data control flow analysis
- Vulnerability detection
- Test cases / fuzzing generation
- Advance monitoring (NSA way)
- Reverse Engineering
- Privacy monitoring
Uses of DBI in Computer Security (II)

Security-related uses

- Data control flow analysis
- Vulnerability detection
- Test cases / fuzzing generation
- Advance monitoring (NSA way)
- Reverse Engineering
- Privacy monitoring
- **Sandboxing**
Uses of DBI in Computer Security (II)

Security-related uses

- Data control flow analysis
- Vulnerability detection
- Test cases / fuzzing generation
- Advance monitoring (NSA way)
- Reverse Engineering
- Privacy monitoring
- Sandboxing
- ...
Uses of DBI in Computer Security (III)

Some security tools that use DBI...

- Vulnerability search
  - SAGE (Microsoft)
  - Sogetis
  - Fuzzgrind
- Avalanche
- Determine
- Pincov
- Taintdroid
- VERA
- TraceSurfer
- ...

J.A. Artal, R.J. Rodríguez, J. Merseguer
Uses of DBI in Computer Security (IV)

Its popularity is *in crescendo* (1)

- **Covert Debugging: Circumventing Software Armoring**, D. Quist & Valsmith, BH USA 2007/DefCon 15
- **Generic Unpacking of Self-modifying, Aggressive, Packed Binary Programs** (P. Bania, CoRR abs/0905.4581 2009)
- **Dynamic Binary Instrumentation for Deobfuscation and Unpacking** (J-Y. Marion & D. Reynaud, DeepSec 2009)
- **Dumping Shellcode with Pin** (S. Porst, Zynamics 2010)
- **Binary Instrumentation for Security Professionals** (G. Diskin, BH USA 2011)
- **Shellcode Analysis using Dynamic Binary Instrumentation** (D. Radu & B. Dang, CARO 2011)
Uses of DBI in Computer Security (V)

Its popularity is *in crescendo* (2)

- **Hacking using Dynamic Binary Instrumentation** (G. Diskin, HITB 2012 AMS)
- **Improving Unpacking Process using DBI techniques** (R.J. Rodríguez, RootedCON 2012)
- **Improving Software Security with Dynamic Binary Instrumentation** (R. Johnson, InfoSec Southwest 2012)
- **Vulnerability Analysis and Practical Data Flow Analysis & Visualization** (J.W. Oh, CanSecWest 2012)
- **Light and Dark side of Code Instrumentation** (D. Evdokimov, CONFidence 2012)
- **Dynamic Binary Instrumentation Frameworks: I know you’re there spying on me** (F. Falcon & N. Riva, RECon 2012)
Outline

1. An Introduction to DBI
   - What (the hell) is Dynamic Binary Instrumentation (DBI)?
   - How does DBI work?
   - Uses of DBI in Computer Security

2. DBI Frameworks
   - DBI Framework: What is?
   - Types of DBI frameworks
   - Analysis and Comparative

3. Applying DBI to Computer Security...
   - Developing DBAs with Pin: Pintools
   - DBI vulnerability search
   - Taint analysis
   - Reverse Engineering

4. Conclusions and Acknowledgments
DBI Framework: What is? (I)

- Provide a bunch of APIs for tool development
- **DBA**: Dynamic Binary Analysis tool
- **DBAs types:**
  - Light-weight
  - Heavy-weight (the use intermediate code)
DBI Framework: What is? (I)

- **Provide a bunch of APIs** for tool development
- **DBA**: Dynamic Binary Analysis tool
- **DBAs types:**
  - Light-weight
  - Heavy-weight (the use intermediate code)
- **Main components**
  - **Core**: just-in-time (JIT) compiler
    - Controls execution of a binary
  - **Library** (this is your own developed tool)
    - Where?
    - What?

\[
\text{$ < DBI\_fw\_core > < myLibrary > < binaryToInstrument >$}
\]
Use modes (most common)

- **JIT**
  - Modification of a (small) set of instructions before executing them
  - More robust
  - Good way for repetitive behaviour binaries (e.g., loops)

- **Probe**
  - Memory patching
  - Less overhead (it executes native code)
  - Not supported by all DBI fws.
DBI Framework: What is? (II)

Use modes (most common)

- **JIT**
  - Modification of a (small) set of instructions before executing them
  - More robust
  - Good way for repetitive behaviour binaries (e.g., loops)

- **Probe**
  - Memory patching
  - Less overhead (it executes native code)
  - Not supported by all DBI fws.

Granularity

<table>
<thead>
<tr>
<th>Instruction Block</th>
<th>Basic Block</th>
<th>Superblock</th>
<th>Trace</th>
<th>Routine</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td></td>
<td>++</td>
<td></td>
<td></td>
<td>--</td>
</tr>
</tbody>
</table>
## DBI Framework: What is? (II)

### Use modes (most common)

- **JIT**
  - Modification of a (small) set of instructions before executing them
  - More robust
  - Good way for repetitive behaviour binaries (e.g., loops)

- **Probe**
  - Memory patching
  - Less overhead (it executes native code)
  - **Not supported by all DBI fws.**

### Granularity

<table>
<thead>
<tr>
<th>Instruction Block</th>
<th>Basic Block</th>
<th>Superblock</th>
<th>Trace Routine</th>
<th>Image</th>
</tr>
</thead>
<tbody>
<tr>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

→ Some not supported in some DBI fws....
## Types of DBI frameworks

### DB fws *in the wild*

<table>
<thead>
<tr>
<th>Pin</th>
<th>DynInst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valgrind</td>
<td>Dtrace</td>
</tr>
<tr>
<td>DynamoRIO</td>
<td>Systemap</td>
</tr>
<tr>
<td>HDtrans</td>
<td></td>
</tr>
</tbody>
</table>
### Types of DBI frameworks (I)

#### DB fws *in the wild*

<table>
<thead>
<tr>
<th>Pin</th>
<th>DynInst</th>
<th>HDtrans</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valgrind</td>
<td>Dtrace</td>
<td></td>
</tr>
<tr>
<td>DynamoRIO</td>
<td>Systemap</td>
<td></td>
</tr>
</tbody>
</table>

Mmm... what is the *much* better?
Types of DBI frameworks (I)

**DB fws in the wild**

<table>
<thead>
<tr>
<th>Pin</th>
<th>DynInst</th>
<th>HDtrans</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valgrind</td>
<td>Dtrace</td>
<td></td>
</tr>
<tr>
<td>DynamoRIO</td>
<td>Systemap</td>
<td></td>
</tr>
</tbody>
</table>

Mmm... what is the *much* better?

**Selection criteria**

- Software being maintained
- License gives access to the source code
- Free
- API provided
- O.S. and common infrastructure
Types of DBI frameworks (II)
Differences y similarities

**Characteristics**
- Source code available (GNU GPL v2)
- Heavy-weight DBAs (using VEX IR as intermediate code)
- [http://www.valgrind.org](http://www.valgrind.org)

**Instruction Basic block Superblock Trace Routine IMage**

<table>
<thead>
<tr>
<th>Framework</th>
<th>Version</th>
<th>Supported Arch.</th>
<th>O.S.</th>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valgrind</td>
<td>3.8.1 (18/09/2012)</td>
<td>Arm, PowerPC, s390, x86, x64</td>
<td>Android, OSX, Linux</td>
<td>IS</td>
</tr>
</tbody>
</table>
## Types of DBI frameworks (II)

Differences y similarities

### Characteristics

- Intel
- Source code available (but proprietary license)
- It allows to attach a process in execution
- [http://www.pintool.org/](http://www.pintool.org/)

### Frameworks

<table>
<thead>
<tr>
<th>Framework</th>
<th>Version</th>
<th>Supported Arch.</th>
<th>O.S.</th>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valgrind</td>
<td>3.8.1 (18/09/2012)</td>
<td>Arm, PowerPC, s390, x86, x64</td>
<td>Android, OSX, Linux</td>
<td>IS</td>
</tr>
<tr>
<td>Pin</td>
<td>2.12 (10/10/2012)</td>
<td>Arm, IA-64, x86, x64</td>
<td>Windows, Linux</td>
<td>IBTRM</td>
</tr>
</tbody>
</table>

Instruction Basic block Superblock Trace Routine IMage
Types of DBI frameworks (II)

Differences y similarities

- MIT, HP, Google
- Source code available (BSD-2)
- Really good docs

### Characteristics

- Instruction
- Basic block
- Superblock
- Trace
- Routine
- IMage

<table>
<thead>
<tr>
<th>Framework</th>
<th>Version</th>
<th>Supported Arch.</th>
<th>O.S.</th>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valgrind</td>
<td>3.8.1 (18/09/2012)</td>
<td>Arm, PowerPC, s390, x86, x64</td>
<td>Android, OSX, Linux</td>
<td>IS</td>
</tr>
<tr>
<td>Pin</td>
<td>2.12 (10/10/2012)</td>
<td>Arm, IA-64, x86, x64</td>
<td>Windows, Linux</td>
<td>I B T R M</td>
</tr>
<tr>
<td>DynamoRIO</td>
<td>3.2.0-3 (01/03/2012)</td>
<td>x86, x64</td>
<td>Windows, Linux</td>
<td>I B T</td>
</tr>
</tbody>
</table>
Types of DBI frameworks (II)
Differences y similarities

Similarities
- Injected code in C/C++
- No need of having the source code of binary to be instrumented
- GNU/Linux x86

Instruction Basic block Superblock Trace Routine IMage

<table>
<thead>
<tr>
<th>Framework</th>
<th>Version</th>
<th>Supported Arch.</th>
<th>O.S.</th>
<th>Granularity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valgrind</td>
<td>3.8.1 (18/09/2012)</td>
<td>Arm, PowerPC, s390, x86, x64</td>
<td>Android, OSX, Linux</td>
<td>IS</td>
</tr>
<tr>
<td>Pin</td>
<td>2.12 (10/10/2012)</td>
<td>Arm, IA-64, x86, x64</td>
<td>Windows, Linux</td>
<td>IBTRM</td>
</tr>
<tr>
<td>DynamoRIO</td>
<td>3.2.0-3 (01/03/2012)</td>
<td>x86, x64</td>
<td>Windows, Linux</td>
<td>IBT</td>
</tr>
</tbody>
</table>
### DBA tool for comparative

- **Counting executed instructions**
- **Two granularities:** instruction and basic block
DBI frameworks comparative (I)

DBA tool for comparative
- Counting executed instructions
- Two granularities: instruction and basic block

Comparative Aim
- Evaluate the performance of selected DBI fws.
- Slowdown: $\frac{t_{\text{instrumented}}}{t_{\text{no\_instrumented}}}$
DBI frameworks comparative (I)

DBA tool for comparative

- Counting executed instructions
- Two granularities: instruction and basic block

Comparative Aim

- Evaluate the performance of selected DBI fws.
  - Slowdown: \( \frac{t_{\text{instrumented}}}{t_{\text{no-instrumented}}} \)

Diving into the APIs

- Pin: ↑ Documentation, ↑↑ Examples, ↑ Tutorials
- DynamoRIO: ↑↑ Documentation, ↑ Examples, ↑ Tutorials
- Valgrind: ↓ Documentation, ↓ Examples, ↓ Tutorials
Experimental settings

- **Hardware**
  - Intel Core2 Duo 2GHz 667MHz, 2GiB DDR2, HDD 120GB

- **Software**
  - Fedora Core 14 32bits, gcc 4.5.1, GNU Fortran 4.5.1, r3
## Experimental settings

- **Hardware**
  - Intel Core2 Duo 2GHz 667MHz, 2GiB DDR2, HDD 120GB

- **Software**
  - Fedora Core 14 32bits, gcc 4.5.1, GNU Fortran 4.5.1, r3

## Benchmark

- Own benchmark created for the comparative
- Considered benchmarks (e.g., SPEC) discarded
- Different categories:
  - Integer computation
  - Float computation
  - I/O
  - Use of memory
Average of memory consumption

![Graph showing memory consumption comparison]
DBI frameworks comparative (III): Results (2)

*Slowdown* by instrumentations
Conclusions
✓ Running optimised code or (int/float) computation → DynamoRIO
✗ Slower solution → Valgrind
  • Memory consumption
    ✓ ↓ Pin
    ✗ ↑ DynamoRIO

Some funny things discovered during the research...
  • No. of instructions differs among the DBI fws. → each one starts in a different point
  • Bug detected when 80-bit numbers rounding in 32 and 64 bits archs. (Valgrind)
    • Already reported:( (https://bugs.kde.org/show_bug.cgi?id=19791)
DBI frameworks comparative (III): Results (4)

Technical Report

- Estudio comparativo de frameworks de Instrumentación Dinámica de Ejecutables (J.A. Artal)
  - For Spanish guys... (we should write some paper soon on this)

Outline

1. An Introduction to DBI
   - What (the hell) is Dynamic Binary Instrumentation (DBI)?
   - How does DBI work?
   - Uses of DBI in Computer Security

2. DBI Frameworks
   - DBI Framework: What is?
   - Types of DBI frameworks
   - Analysis and Comparative

3. Applying DBI to Computer Security...
   - Developing DBAs with Pin: Pintools
   - DBI vulnerability search
   - Taint analysis
   - Reverse Engineering

4. Conclusions and Acknowledgments
Developing DBAs with Pin: Pintools (I)

- VM + code cache + API instrumentation
- DBA $\rightarrow$ Pintool
- VM: JIT + emulator + dispatcher
Developing DBAs with Pin: Pintools (I)

- VM + code cache + API instrumentation
- DBA → Pintool
- VM: JIT + emulator + dispatcher
  1. JIT compiles and instruments the binary code
  2. Launched by the dispatcher
  3. Stored in code cache
Developing DBAs with Pin: Pintools (I)

- VM + code cache + API instrumentation
- DBA → Pintool
- VM: JIT + emulator + dispatcher
  1. JIT compiles and instruments the binary code
  2. Launched by the dispatcher
  3. Stored in code cache
- Works on the O.S.: user-space
Developing DBAs with Pin: Pintools (II)

An example: inscount.cpp

```c++
#include "pin.H"

//Instruction counter
static UINT64 icount = 0;

// Called before every instruction is executed
VOID docount() { icount++; }

// Called every time a new instruction is encountered
VOID Instruction(INS ins, VOID *v){
    // Insert a call to docount before every instruction, no arguments are passed
    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)docount, IARG_END);
}

// Called when the application exits
VOID Fini(INT32 code, VOID *v){
    std::cout << "Count " << icount << endl;
}

int main(int argc, char * argv[]){
    PIN_Init(argc, argv);
    INS_AddInstrumentFunction(Instruction, 0);
    PIN_AddFiniFunction(Fini, 0);
    PIN_StartProgram(); // no returns
    return 0;
}
```
DBI vulnerability search (I): Double Free

Demo: DoubleFreeDBA.dll

Vulnerability description

- **CWE-415** ([http://cwe.mitre.org/data/definitions/415.html](http://cwe.mitre.org/data/definitions/415.html))
- Call `free()` with the same `@` → **corrupt memory**
- "Doubly freeing memory may result in a write-what-where condition, allowing an attacker to execute arbitrary code"

DBA developed with Pin (DoubleFreeDBA.dll)

- **Where?**
  - APIs `RtlAllocateHeap` (after), `RtlAllocateFree` (before)
- **What?**
  - `RtlAllocateHeap`: keeps returned `@` in a list
  - `RtlAllocateFree`: removes `@` from list, and reports if not found!
Vulnerability description

- **CWE-415** ([http://cwe.mitre.org/data/definitions/415.html](http://cwe.mitre.org/data/definitions/415.html))
- Call `free()` with the same @ → corrupt memory
- “Doubly freeing memory may result in a write-what-where condition, allowing an attacker to execute arbitrary code”

DBA developed with Pin (DoubleFreeDBA.dll)

- **Where?**
  - APIs `RtlAllocateHeap` (after), `RtlAllocateFree` (before)
- **What?**
  - `RtlAllocateHeap`: keeps returned @ in a list
  - `RtlAllocateFree`: removes @ from list, and reports if not found!

**Friendly reminder**: Make a demo...
DBI vulnerability search (II): Buffer Overflow (1)
Demo: BufferOverflowDBA.dll

Vulnerability description

- **CWE-120** ([http://cwe.mitre.org/data/definitions/120.html](http://cwe.mitre.org/data/definitions/120.html))
- Copy a buffer without restrictions → **arbitrary code execution**
- "Buffer overflows often can be used to execute arbitrary code [...]. Buffer overflows generally lead to crashes [...]."

DBA developed with Pin (BufferOverflowDBA.dll)

- Works around **scanf**
- **Where?** → API **scanf** (before)
- **What?**
  - Checks parameters seeking buffers without limits
- **Improvements**: extend to other vulnerable APIs (e.g., `strcpy`)
DBI vulnerability search (II): Buffer Overflow (1)

Demo: BufferOverflowDBA.dll

Vulnerability description

- CWE-120 ([http://cwe.mitre.org/data/definitions/120.html](http://cwe.mitre.org/data/definitions/120.html))
- Copy a buffer without restrictions → arbitrary code execution
- “Buffer overflows often can be used to execute arbitrary code [...]. Buffer overflows generally lead to crashes [...].”

DBA developed with Pin (BufferOverflowDBA.dll)

- Works around `scanf`
- Where? → API `scanf` (before)
- What?
  - Checks parameters seeking buffers without limits
- Improvements: extend to other vulnerable APIs (e.g., `strcpy`)

Friendly reminder: Make a demo...
DBI vulnerability search (II): Buffer Overflow (2)

Demo: ProtectRetAddrDBA.dll

Vulnerability description

- **CWE-120** ([http://cwe.mitre.org/data/definitions/120.html](http://cwe.mitre.org/data/definitions/120.html))
- Copy a buffer without restrictions → **arbitrary code execution**
- “Buffer overflows often can be used to execute arbitrary code [...]. Buffer overflows generally lead to crashes [...]”

DBA developed with Pin (ProtectRetAddrDBA.dll)

- **Where?** → every CALL (before) o RETN (before) in .text section
- **What?**
  - CALL → stores legitimate return address \((EIP + \text{size}(CALL))\)
  - RETN → checks if retn address is in the list...
- Detected 6 retn changes in ntdll.dll library!!
DBI vulnerability search (II): Buffer Overflow (2)

Demo: ProtectRetAddrDBA.dll

Vulnerability description

- **CWE-120** ([http://cwe.mitre.org/data/definitions/120.html](http://cwe.mitre.org/data/definitions/120.html))
- Copy a buffer without restrictions → **arbitrary code execution**
- "Buffer overflows often can be used to execute arbitrary code [...]. Buffer overflows generally lead to crashes [...]."

DBA developed with Pin ([ProtectRetAddrDBA.dll](http://cwe.mitre.org/data/definitions/120.html))

- **Where?** → every CALL (before) or RETN (before) in .text section
- **What?**
  - CALL → stores legitimate return address \((EIP + \text{size}(\text{CALL}))\)
  - RETN → checks if retn address is in the list...
- Detected 6 retn changes in ntdll.dll library!!

**Friendly reminder:** Make a demo...
DBI vulnerability search (III): Taint analysis

Demo: TaintAnalysisDBA.dll

DBA developed with Pin (TaintAnalysisDBA.dll)

- Taint analysis of `scanf` parameters
- Where? → API `scanf` (after)
- What?
  - Trace all registers/memory zones contaminated from the input data
DBI vulnerability search (III): Taint analysis

Demo: TaintAnalysisDBA.dll

DBA developed with Pin (TaintAnalysisDBA.dll)

- Taint analysis of `scanf` parameters
- Where? → API `scanf` (after)
- What?
  - Trace all registers/memory zones contaminated from the input data

Friendly reminder: Make a demo...
DBI vulnerability search (IV): Reverse Engineering

Demo: EasyPasswordDBA.dll – very naif example

DBA developed with Pin (EasyPasswordDBA.dll)

- Seeking for the correct key
- **Hook** when calling to string comparison `lstrcmpA`
- Where?
  - API `lstrcmpA` (before)
- What?
  - Log all function parameters
**DBI vulnerability search (IV): Reverse Engineering**

Demo: EasyPasswordDBA.dll – very naif example

---

**DBA developed with Pin (EasyPasswordDBA.dll)**

- Seeking for the correct key
- **Hook** when calling to string comparison `lstrcmpA`
- **Where?**
  - API `lstrcmpA` (before)
- **What?**
  - Log all function parameters
- This is not longer valid for current apps... isn’t it? 😊
DBI vulnerability search (IV): Reverse Engineering
Demo: EasyPasswordDBA.dll – very naif example

DBA developed with Pin (EasyPasswordDBA.dll)

- Seeking for the correct key
- *Hook* when calling to string comparison `lstrcmpA`
- Where?
  - API `lstrcmpA` (before)
- What?
  - Log all function parameters
- This is not longer valid for current apps... *isn’t it? 😊*

*Friendly reminder:* Make a demo...
Outline

1. An Introduction to DBI
   - What (the hell) is Dynamic Binary Instrumentation (DBI)?
   - How does DBI work?
   - Uses of DBI in Computer Security

2. DBI Frameworks
   - DBI Framework: What is?
   - Types of DBI frameworks
   - Analysis and Comparative

3. Applying DBI to Computer Security...
   - Developing DBAs with Pin: Pintools
   - DBI vulnerability search
   - Taint analysis
   - Reverse Engineering

4. Conclusions and Acknowledgments
Conclusions

- DBI frameworks: fast and easy development → high potential
- NO need of (too much) advanced O.S. programming knowledge
  - We can focus in what really matters: our DBA tool
- Disadvantages:
  - DBI API knowledge
  - Execution time

Recall about the DBI fws. comparison...

✓ Running optimised code or (int/float) computation → DynamoRIO
✗ Slower solution → Valgrind
- Memory consumption
  ✓ ↓ Pin
 ✗ ↑ DynamoRIO
Acknowledgments

- Gal Diskin
- Dimitry “D1g1” Evdokimov
- Francisco Falcon & Nahuel Riva
- CrackLatinoS (CLS)
- Hack in Paris staff, thank you guys & gals!
Acknowledgments

- Gal Diskin
- Dimitry “D1g1” Evdokimov
- Francisco Falcon & Nahuel Riva
- CrackLatinoS (CLS)
- Hack in Paris staff, thank you guys & gals!
- To you for hearing me stoically...
Juan Antonio Artal‡, Ricardo J. Rodríguez†, José Merseguer‡

© All wrongs reversed
jaartal@gmail.com, rjrodriguez@fi.upm.es, jmerse@unizar.es
@RicardoJRdez ※ www.ricardojrodriguez.es

†Universidad Politécnica de Madrid
Madrid, Spain

‡ Universidad de Zaragoza
Zaragoza, Spain

June 21th, 2013

3rd Edition of Hack in Paris
Sequoia Lodge Hotel, Disneyland Paris