esiea
ECOLE D'INGENIEURS
DU MONDE NUMERIQUE

Introduction à PIN, un framework de DBI

Sommaire

- Whoami
- Un framework DBI, C'est quoi?
- PIN de Intel
- Exemple : le crackme
- Exemple : la résolution
- Conclusion

Whoami

- Patrick Ventuzelo
- @Pat_Ventuzelo
- Etudiant au MSSIS de l'ESIEA

- Passionné par :
  - Reverse, Exploit, Android, Python …
  - Challenge (Rootme, CTF, …)

Un framework DBI, c'est quoi?

Dynamic Binary Instrumentation

- Analyse et modification en Runtime
- Pas d'altération du binaire
- Pas de recompilation
- Instrumente également le code généré dynamiquement

A quoi ça sert? (InfoSec)

- Reverse
- Code coverage / Fuzzing
- Detection de vulnérabilité
- Pré-patch de vulnérabilité
- Tainting
- Analyse de malware
- Unpacking
- ...

Quel framework DBI choisir?

- PIN (Linux, Windows, OSX, Android)
  - https://software.intel.com/en-us/articles/pintool-downloads
- DynamoRIO (Linux, Windows, Android)
  - http://www.dynamorio.org/
- Valgrind (Linux, OSX, Android)
  - http://valgrind.org/

PIN d' Intel

PIN d' Intel

- Simple d'utilisation et d'implémentation (Pintool)
- Les Pintools sont codés en C/C++ ou Python (binding)
- Une API et une doc complète
- Multi-plateforme
  - x86, x86-64, …
  - Linux, Windows, OSX, Android
- Pas besoin des sources du binaire
- Pas besoin de recompiler le binaire
- Fonctionne sur des Applis lourdes (multithreadé, …)
- Moteur de debug intégré et intégrable

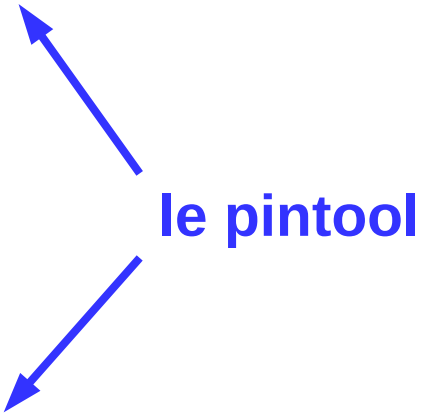
./pin --help

- pin -t inscount0.so -- pwd

- pin -t itrace.so -pid 1337

./pin --help

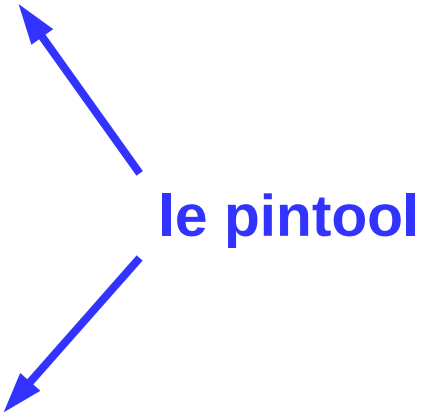
- pin -t inscount0.so -- pwd



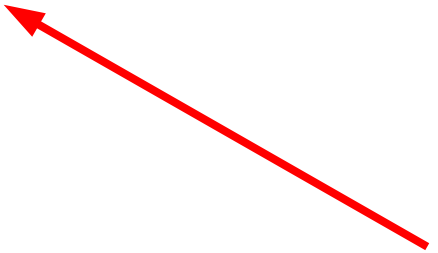
- pin -t itrace.so -pid 1337

./pin --help

- pin -t inscount0.so -- pwd



le pintool

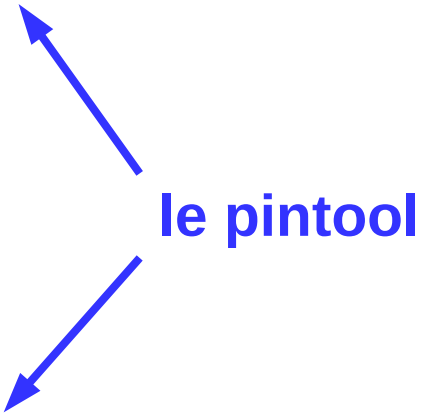


l'application à exécuter

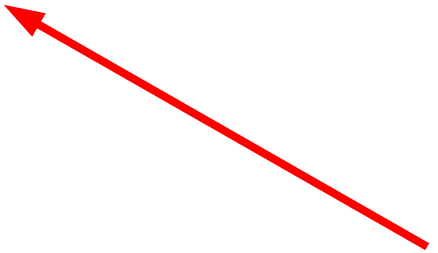
- pin -t itrace.so -pid 1337

./pin --help

- pin -t inscount0.so -- pwd

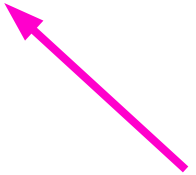


le pintool



l'application à exécuter

- pin -t itrace.so -pid 1337



pid de l'application déjà lancé

Pintool inscount0

- Principe :

```
counter++;
sub $0xff, %edx
counter++;
cmp %esi, %edx
counter++;
jle <L1>
counter++;
mov $0x1, %edi
counter++;
add $0x10, %eax
```

Code de inscount0.cpp - main

```
80  int main(int argc, char * argv[])
81  {
82      // Initialize pin
83      if (PIN_Init(argc, argv)) return Usage();
84
85      OutFile.open(KnobOutputFile.Value().c_str());
86
87      // Register Instruction to be called to instrument instructions
88      INS_AddInstrumentFunction(Instruction, 0);
89
90      // Register Fini to be called when the application exits
91      PIN_AddFiniFunction(Fini, 0);
92
93      // Start the program, never returns
94      PIN_StartProgram();
95
96      return 0;
97  }
```

Code de inscount0.cpp - Instruction

```
41  // This function is called before every instruction is executed
42  VOID docount() { icount++; }
43
44  // Pin calls this function every time a new instruction is encountered
45  VOID Instruction(INS ins, VOID *v)
46  {
47      // Insert a call to docount before every instruction, no arguments are passed
48      INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)docount, IARG_END);
49  }
```

Code de inscount0.cpp - Instruction

```
41  // This function is called before every instruction is executed
42  VOID docount() { icount++; }
43
44  // Pin calls this function every time a new instruction is encountered
45  VOID Instruction(INS ins, VOID *v)
46  {
47      // Insert a call to docount before every instruction, no arguments are passed
48      INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)docount, IARG_END);
49  }
```

VOID LEVEL_PINCLIENT::INS_InsertCall(INS *ins*,
IPOINT *action*,
AFUNPTR *funptr*,
...
)

Insert a call to funptr relative to instruction ins.

Parameters:

ins	Instruction to instrument
action	Specifies before, after, etc. IPOINT_BEFORE is always valid for all instructions. IPOINT_AFTER is valid only when a fall-through exists (i.e. Calls and unconditional branches will fail). IPOINT_TAKEN_BRANCH is invalid for non-branches.
funptr	Insert a call to funptr
...	List of arguments to pass funptr. See **IARG_TYPE**, terminated with IARG_END

If more than one call is inserted for the same instruction, the order is determined by **IARG_CALL_ORDER**. For more information, see **CALL_ORDER**.

Code de inscount0.cpp - Instruction

```
41  // This function is called before every instruction is executed
42  VOID docount() { icount++; }
43
44  // Pin calls this function every time a new instruction is encountered
45  VOID Instruction(INS ins, VOID *v)
46  {
47      // Insert a call to docount before every instruction, no arguments are passed
48      INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)docount, IARG_END);
49  }
```

VOID LEVEL_PINCLIENT::INS_InsertCall(INS *ins*, IPOINT *action*, AFUNPTR *funptr*, ...)

Insert a call to funptr relative to instruction ins.

Parameters:

ins	Instruction to instrument
action	Specifies before, after, etc. IPOINT_BEFORE is always valid for all instructions. IPOINT_AFTER is valid only when a fall-through exists (i.e. Calls and unconditional branches will fail). IPOINT_TAKEN_BRANCH is invalid for non-branches.
funptr	Insert a call to funptr
...	List of arguments to pass funptr. See **IARG_TYPE**, terminated with IARG_END

If more than one call is inserted for the same instruction, the order is determined by **IARG_CALL_ORDER**. For more information, see **CALL_ORDER**.

Code de inscount0.cpp - Instruction

```
41  // This function is called before every instruction is executed
42  VOID docount() { icount++; }
43
44  // Pin calls this function every time a new instruction is encountered
45  VOID Instruction(INS ins, VOID *v)
46  {
47      // Insert a call to docount before every instruction, no arguments are passed
48      INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)docount, IARG_END);
49  }
```

VOID LEVEL_PINCLIENT::INS_InsertCall(INS *ins*,

enum IPOINT

Determines where the analysis call is inserted relative to the instrumented object

Enumerator:

IPOINT_BEFORE	Insert a call before an instruction or routine.
IPOINT_AFTER	Insert a call on the fall thorugh path of an instruction or return path of a routine.
IPOINT_ANYWHERE	Insert a call anywhere inside a trace or a bbl.
IPOINT_TAKEN_BRANCH	Insert a call on the taken edge of branch, the side effects of the branch are visible.

... List of arguments to pass funptr. See **IARG_TYPE**, terminated with IARG_END

If more than one call is inserted for the same instruction, the order is determined by **IARG_CALL_ORDER**. For more information, see **CALL_ORDER**.

Code de inscount0.cpp - Instruction

```
41  // This function is called before every instruction is executed
42  VOID docount() { icount++; }
43
44  // Pin calls this function every time a new instruction is encountered
45  VOID Instruction(INS ins, VOID *v)
46  {
47      // Insert a call to docount before every instruction, no arguments are passed
48      INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)docount, IARG_END);
49  }
```

VOID LEVEL_PINCLIENT::INS_InsertCall(INS *ins*,

enum IPOINT

Determines where the analysis call is inserted relative to the instrumented object

Enumerator:

IPOINT_BEFORE	Insert a call before an instruction or routine.
IPOINT_AFTER	Insert a call on the fall thorugh path of an instruction or return path of a routine.
IPOINT_ANYWHERE	Insert a call anywhere inside a trace or a bbl.
IPOINT_TAKEN_BRANCH	Insert a call on the taken edge of branch, the side effects of the branch are visible.

... List of arguments to pass funptr. See **IARG_TYPE**, terminated with IARG_END

If more than one call is inserted for the same instruction, the order is determined by **IARG_CALL_ORDER**. For more information, see **CALL_ORDER**.

Pintool itrace

- Principe :

```
Print(ip);
sub $0xff, %edx
Print(ip);
cmp %esi, %edx
Print(ip);
jle <L1>
Print(ip);
mov $0x1, %edi
Print(ip);
add $0x10, %eax
```

Code de itrace.cpp - main

```
69  int main(int argc, char * argv[])
70  {
71      trace = fopen("itrace.out", "w");
72
73      // Initialize pin
74      if (PIN_Init(argc, argv)) return Usage();
75
76      // Register Instruction to be called to instrument instructions
77      INS_AddInstrumentFunction(Instruction, 0);
78
79      // Register Fini to be called when the application exits
80      PIN_AddFiniFunction(Fini, 0);
81
82      // Start the program, never returns
83      PIN_StartProgram();
84
85      return 0;
86  }
```

Code de itrace.cpp - Instruction

```
38  VOID printip(VOID *ip) { fprintf(trace, "%p\n", ip); }
39
40  // Pin calls this function every time a new instruction is encountered
41  VOID Instruction(INS ins, VOID *v)
42  {
43      // Insert a call to printip before every instruction, and pass it the IP
44      INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)printip, IARG_INST_PTR, IARG_END);
45  }
```

Code de itrace.cpp - Instruction

```
38  VOID printip(VOID *ip) { fprintf(trace, "%p\n", ip); }
39
40  // Pin calls this function every time a new instruction is encountered
41  VOID Instruction(INS ins, VOID *v)
42  {
43      // Insert a call to printip before every instruction, and pass it the IP
44      INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)printip, IARG_INST_PTR, IARG_END);
45  }
```

```
VOID LEVEL_PINCLIENT::INS_InsertCall( INS      ins,
                                      IPOINT   action,
                                      AFUNPTR  funptr,
                                      ...
                                    )
```

Insert a call to funptr relative to instruction ins.

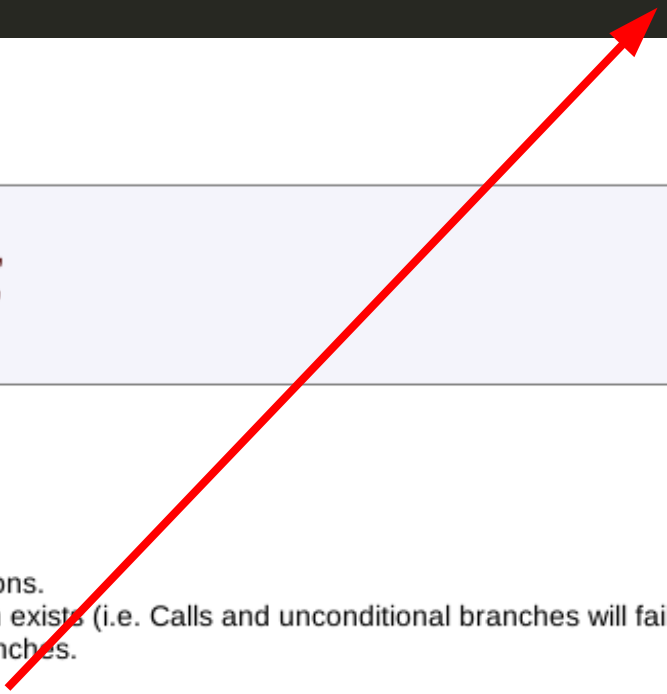
Parameters:

ins	Instruction to instrument
action	Specifies before, after, etc. IPOINT_BEFORE is always valid for all instructions. IPOINT_AFTER is valid only when a fall-through exists (i.e. Calls and unconditional branches will fail). IPOINT_TAKEN_BRANCH is invalid for non-branches.
funptr	Insert a call to funptr
...	List of arguments to pass funptr. See **IARG_TYPE**, terminated with IARG_END

If more than one call is inserted for the same instruction, the order is determined by **IARG_CALL_ORDER**. For more information, see **CALL_ORDER**.

Code de itrace.cpp - Instruction

```
38  VOID printip(VOID *ip) { fprintf(trace, "%p\n", ip); }
39
40  // Pin calls this function every time a new instruction is encountered
41  VOID Instruction(INS ins, VOID *v)
42  {
43      // Insert a call to printip before every instruction, and pass it the IP
44      INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)printip, IARG_INST_PTR, IARG_END);
45  }
```



VOID LEVEL_PINCLIENT::INS_InsertCall(INS *ins*,
IPOINT *action*,
AFUNPTR *funptr*,
...
)

Insert a call to funptr relative to instruction ins.

Parameters:

ins	Instruction to instrument
action	Specifies before, after, etc. IPOINT_BEFORE is always valid for all instructions. IPOINT_AFTER is valid only when a fall-through exists (i.e. Calls and unconditional branches will fail). IPOINT_TAKEN_BRANCH is invalid for non-branches.
funptr	Insert a call to funptr
...	List of arguments to pass funptr. See **IARG_TYPE**, terminated with IARG_END

If more than one call is inserted for the same instruction, the order is determined by **IARG_CALL_ORDER**. For more information, see **CALL_ORDER**.

Code de itrace.cpp - Instruction

```
38  VOID printip(VOID *ip) { fprintf(trace, "%p\n", ip); }
39
40  // Pin calls this function every time a new instruction is encountered
41  VOID Instruction(INS ins, VOID *v)
42  {
43      // Insert a call to printip before every instruction, and pass it the IP
44      INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)printip, IARG_INST_PTR, IARG_END);
45  }
```

VOID LEVEL_PINCLIENT::INS_InsertCall(INS *ins*,
IPOINT *action*,
AFUNPTR *funptr*,
...

enum IARG_TYPE

Determines the arguments that are passed to the analysis call. All argument lists must end with IARG_END.

Enumerator:

IARG_ADDRINT	Type: ADDRINT. Constant value (additional arg required).
IARG_PTR	Type: "VOID *". Constant value (additional pointer arg required).
IARG_BOOL	Type: BOOL. Constant (additional BOOL arg required).
IARG_UINT32	Type: UINT32. Constant (additional integer arg required).
IARG_INST_PTR	Type: ADDRINT. The address of the instrumented instruction.

Code de itrace.cpp - Instruction

```
38  VOID printip(VOID *ip) { fprintf(trace, "%p\n", ip); }
39
40  // Pin calls this function every time a new instruction is encountered
41  VOID Instruction(INS ins, VOID *v)
42  {
43      // Insert a call to printip before every instruction, and pass it the IP
44      INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)printip, IARG_INST_PTR, IARG_END);
45  }
```

VOID LEVEL_PINCLIENT::INS_InsertCall(INS *ins*,
IPOINT *action*,
AFUNPTR *funptr*,
...

enum IARG_TYPE

Determines the arguments that are passed to the analysis call. All argument lists must end with IARG_END.

Enumerator:

IARG_ADDRINT	Type: ADDRINT. Constant value (additional arg required).
IARG_PTR	Type: "VOID *". Constant value (additional pointer arg required).
IARG_BOOL	Type: BOOL. Constant (additional BOOL arg required).
IARG_UINT32	Type: UINT32. Constant (additional integer arg required).
IARG_INST_PTR	Type: ADDRINT. The address of the instrumented instruction.

Exemple : le crackme


haley

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void header(void){...
}

char password[16] =
"\x01\x30\x76\x01\x29\x0f\x27\x1d\x07\x11\x71\x1d\x70\x0d\x2e\x74";
                    /* xor("Cr4CkMe_ES3_2016",0x42) */

int check(char pass, char input){
    if(pass != (input ^ 0x42)){
        printf("[-] Try again ;) \n");
        return 1;
    }
    return 0;
}

int main(int argc, char *argv[])
{
    int offset = 0;

    /* */
    header();
    if(argc != 2){ printf("[-] USAGE : %s <password>\n",argv[0]); return 1;}

    /* Check length user_input*/
    if(strlen(argv[1]) != strlen(password)) return 1;

    for(offset = 0; offset < strlen(password); offset++){
        if(check(password[offset],argv[1][offset]))
            return 1;
    }

    printf("[+] Well play ;) \n");
    return 0;
}
```

Exemple : le crackme

- Taille du password = 16
- Password = xor(“Cr4CkMe_ES3_2Ol6”,0x42)

- Majuscules, minuscules, chiffres, caractères spéciaux

Exemple : la résolution

- Trouver la taille du password

```
 7  ###### find password length ######
 8
 9  import commands
10
11  password_len = 0
12  liste_count = list()
13
14  print "[*] Recherche de la taille du password "
15  for i in xrange(1,20,1):
16      command = "../../../pin.sh -t inscount0.so -o out.txt -- \
17                  ./crackme \"%s\" > /dev/null ; cat out.txt" %('A'*i)
18      result_out = commands.getoutput(command)
19      liste_count.append(int(result_out.split("Count")[1]))
20
21  password_len = liste_count.index(max(liste_count))+1
22  print "[*] Taille du password = " + str(password_len)
```

Exemple : la résolution

- Trouver le password

```
25  ###### find password ######
26
27  print "[*] Recherche du password "
28  password = ""
29  offset = 0
30  charset = " !#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[] \
31                              ^_abcdefghijklmnopqrstuvwxyz{|}"
32  base_count = 0
33  char = " "
34  while offset < password_len :
35      print "[*] Recherche du caractere a l'offset " + str(offset)
36      liste_count = list()
37
38      for i in charset:
39
40          test = password + i + "A"*(password_len-offset-1)
41          command = '../../../pin.sh -t inscount0.so -o out.txt -- \
42                      ./crackme \"%s\" > /dev/null ; cat out.txt' % (test)
43          result_out = commands.getoutput(command)
44          actual_count = int(result_out.split("Count")[1])
45
46          # nombre d instruction 1er essai
47          if i == ' ':
48              base_count = actual_count
49          # check si le nombre d'instruction a augmente
50          if actual_count > base_count:
51              char = i
52              break
53      password += char
54      print "[*] " + password
55      offset += 1
56  print "[*]"
57  print "[*] Password = " + password
58  print "[*]"
59
```

Exemple : résultat

Conclusion

- Les frameworks DBI c'est cool ;)
- L'API de Pin est très complète
- Enormement d'exemples fournis et dispo sur Github

- Essayez-le ;)

Question

PIN vs Crackme

- http://shell-storm.org/blog/A-binary-analysis-count-me-if-you-can/
- http://rmolina.co/2015/10/solucion-automatica-de-crackmes.html
- https://www.aldeid.com/wiki/The-FLARE-On-Challenge-2015/Challenge-9
- http://security.cs.pub.ro/hexcellents/wiki/writeups/codegate2014_dodocrackme
- http://parsiya.net/blog/2014-12-08-pin-adventures---chapter-1---pinsolver-mk1/
- https://sysexit.wordpress.com/2013/09/04/a-black-box-approach-against-obfuscated-regular-expressions-using-pin/

Liens utiles

- http://shell-storm.org/repo/Notepad/more-Pin-stuff-references.txt
- http://software.intel.com/sites/landingpage/pintool/docs/71313/Pin/html/group__API_REF.html
- https://media.blackhat.com/bh-us-11/Diskin/BH_US_11_Diskin_Binary_Instrumentation_Slides.pdf
- https://cs.gmu.edu/~astavrou/courses/ISA_673_S13/PIN_lecture.pdf
- http://www.cs.du.edu/~dconnors/courses/comp3361/notes/PinTutorial
- http://2011.zeronights.org/files/dmitriyd1g1evdokimov-dbiintro-111202045015-phpapp01.pdf
- http://doar-e.github.io/blog/2013/08/31/some-thoughts-about-code-coverage-measurement-with-pin/
- http://resources.infosecinstitute.com/pin-dynamic-binary-instrumentation-framework/
- https://recon.cx/2014/slides/pinpoint_control_for_analyzing_malware_recon2014_jjones.pdf
- https://msdn.microsoft.com/fr-fr/magazine/dn818497.aspx