Light and Dark side of Code Instrumentation

Dmitriy “D1g1” Evdokimov

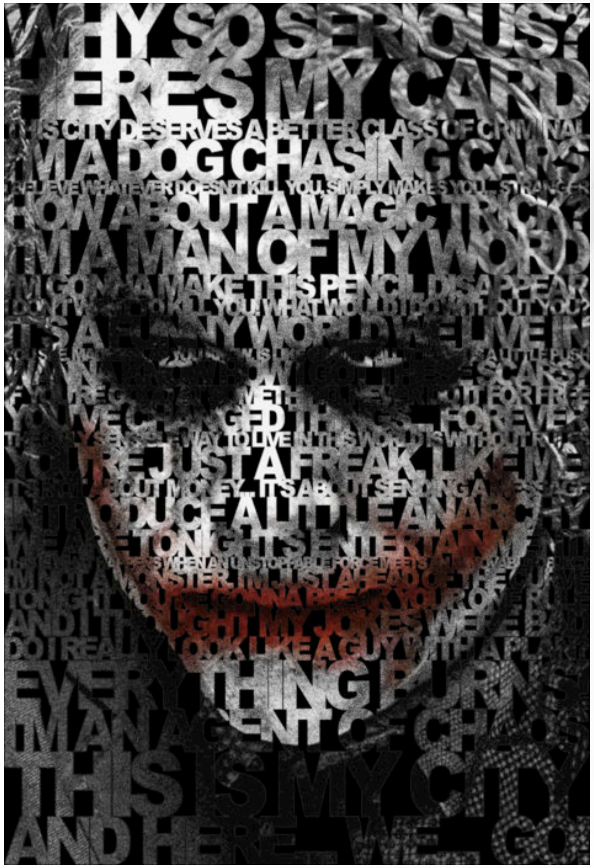
DSecRG, Security Researcher

#whoami

- Security Researcher in DSecRG
  - RE
  - Fuzzing
  - Mobile security
- Organizer: DCG #7812
- Editor in “XAKEP”

Agenda

1. Instrumentation .
2. Instrumentation ..
3. Instrumentation …
4. Instrumentation ….
5. Instrumentation …..
6. Instrumentation ……
7. Instrumentation …….

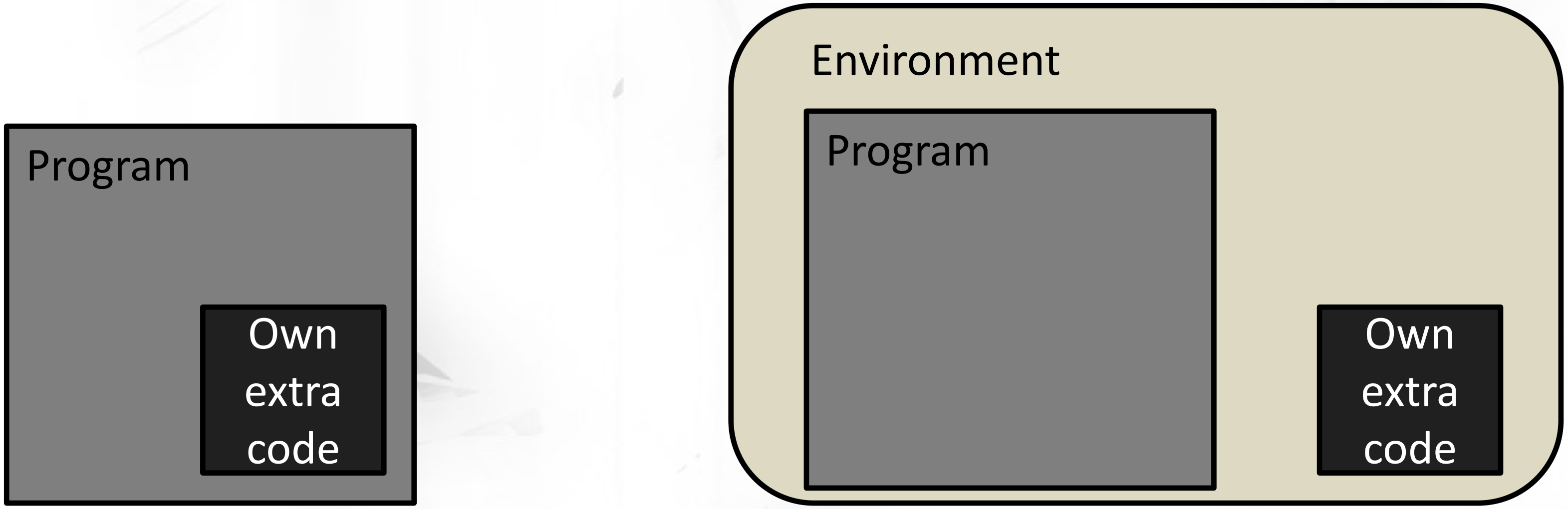


Intro

"It has been proved by scientists that a new point of evolution, any technical progress appears when a Man makes up a new type of tool, but not a product."

Instrumentation

Instrumentation is a technique adding extra code to an program/environment for monitoring/change some program behavior.



Why is it necessary?

Parallel optimization

Simulation

Memory debugging

Virtualization

Emulation

Software profiling

Performance analysis

Testing

Automated debugging

Correctness checking

Error detection

Collecting code metrics

Binary translation

Optimization

Memory leak detection

Instrumentation in information security

Control flow analysis

Security test case generation

Unpack

Code coverage

Virtual patching

Malware analysis

Vulnerability detection

Privacy monitoring

Taint analysis

Antivirus technology

Program shepherding

Sandboxing

Data flow analysis

Shellcode detection

Fuzzing

Deobfuscation

Reverse engineering

Behavior based security

Forensic

Transparent debugging

Data Structure Restoring

Security enforcement

Analysis

Criterion	Static analysis	Dynamic analysis
Code vs. data	Problem	No problem
Code coverage	Big (but not all)	One way
Information about values	No information	All information
Self-modifying code	Problem	No problem
Interaction with the environment	No	Yes
Unused code	Analysis	No analysis
JIT code	Problem	No problem

Code Discovery

Memory

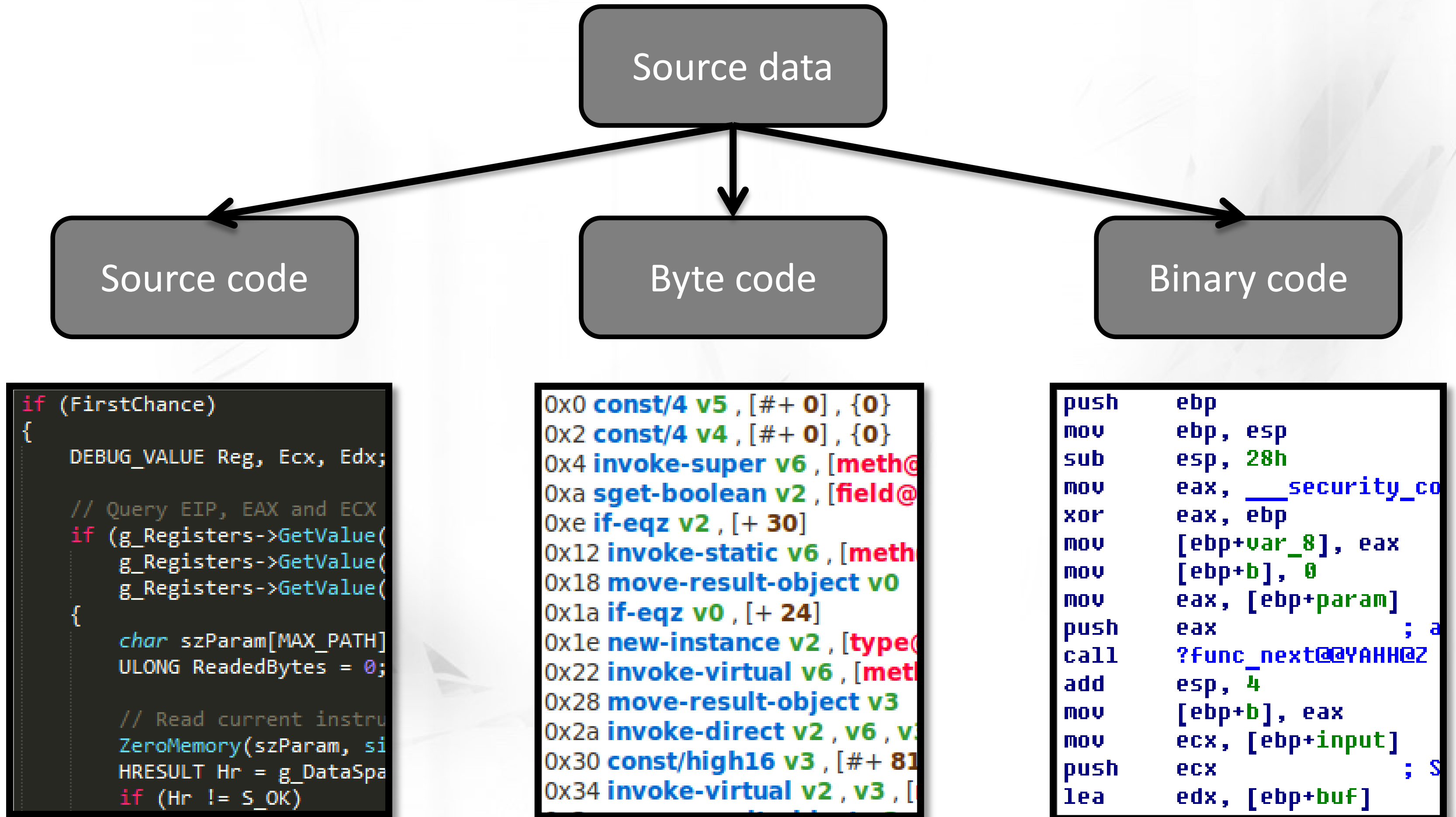
After static analysis

After dynamic analysis

The general scheme of code instrumentation

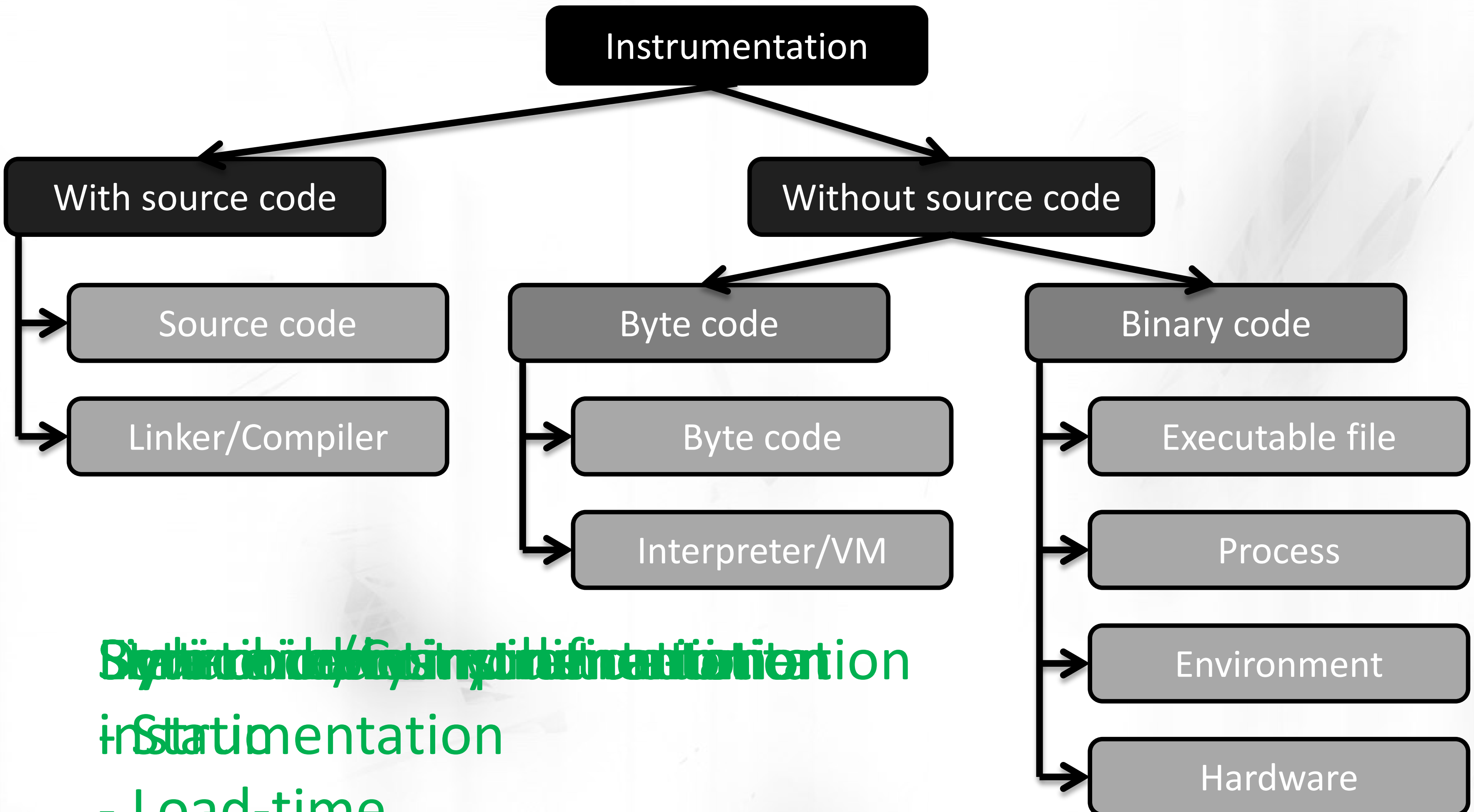
1. Find points of instrumentation;
2. Insert instrumentation;
3. Take control from program;
4. Save context of the program;
5. Execute own code;
6. Restore context of the program;
7. Return control to program.

Source Data



Source data
Source code
Byte code
Binary code
if (FirstChance)
{
DEBUG_VALUE Reg, Ecx, Edx;
// Query EIP, EAX and ECX
if (g_Registers->GetValue(
g_Registers->GetValue(
g_Registers->GetValue(
{
char szParam[MAX_PATH]
ULONG ReadedBytes = 0;
// Read current instru
ZeroMemory(szParam, si
HRESULT Hr = g_DataSpa
if (Hr != S_OK)
0x0 const/4 v5 , [#+ 0] , {0}
0x2 const/4 v4 , [#+ 0] , {0}
0x4 invoke-super v6 , [meth@
0xa sget-boolean v2 , [field@
0xe if-eqz v2 , [+ 30]
0x12 invoke-static v6 , [meth
0x18 move-result-object v0
0x1a if-eqz v0 , [+ 24]
0x1e new-instance v2 , [type@
0x22 invoke-virtual v6 , [met
0x28 move-result-object v3
0x2a invoke-direct v2 , v6 , v
0x30 const/high16 v3 , [#+ 81
0x34 invoke-virtual v2 , v3 , [
push ebp
mov ebp, esp
sub esp, 28h
mov eax, ___security_co
xor eax, ebp
mov [ebp+var_8], eax
mov [ebp+b], 0
mov eax, [ebp+param]
push eax ; a
call ?func_next@@YAHH@Z
add esp, 4
mov [ebp+b], eax
mov ecx, [ebp+input]
push ecx ; S
lea edx, [ebp+buf]

Classification of target instrumentation



- Static
- Load-time
- Dynamic

Source code instrumentation

- Source code*
  - Source code instrumentation
    - Manual skills
    - Plugins for IDE
  - Link-time/Compilation-time instrumentation
    - Options of linker/compiler
- Tools: Visual Studio Profiler, gcc, TAU, OPARI, Diablo, Phoenix, LLVM, Rational Purify, Valgrind

*Unreal condition for security specialist =)

Unmoral programming

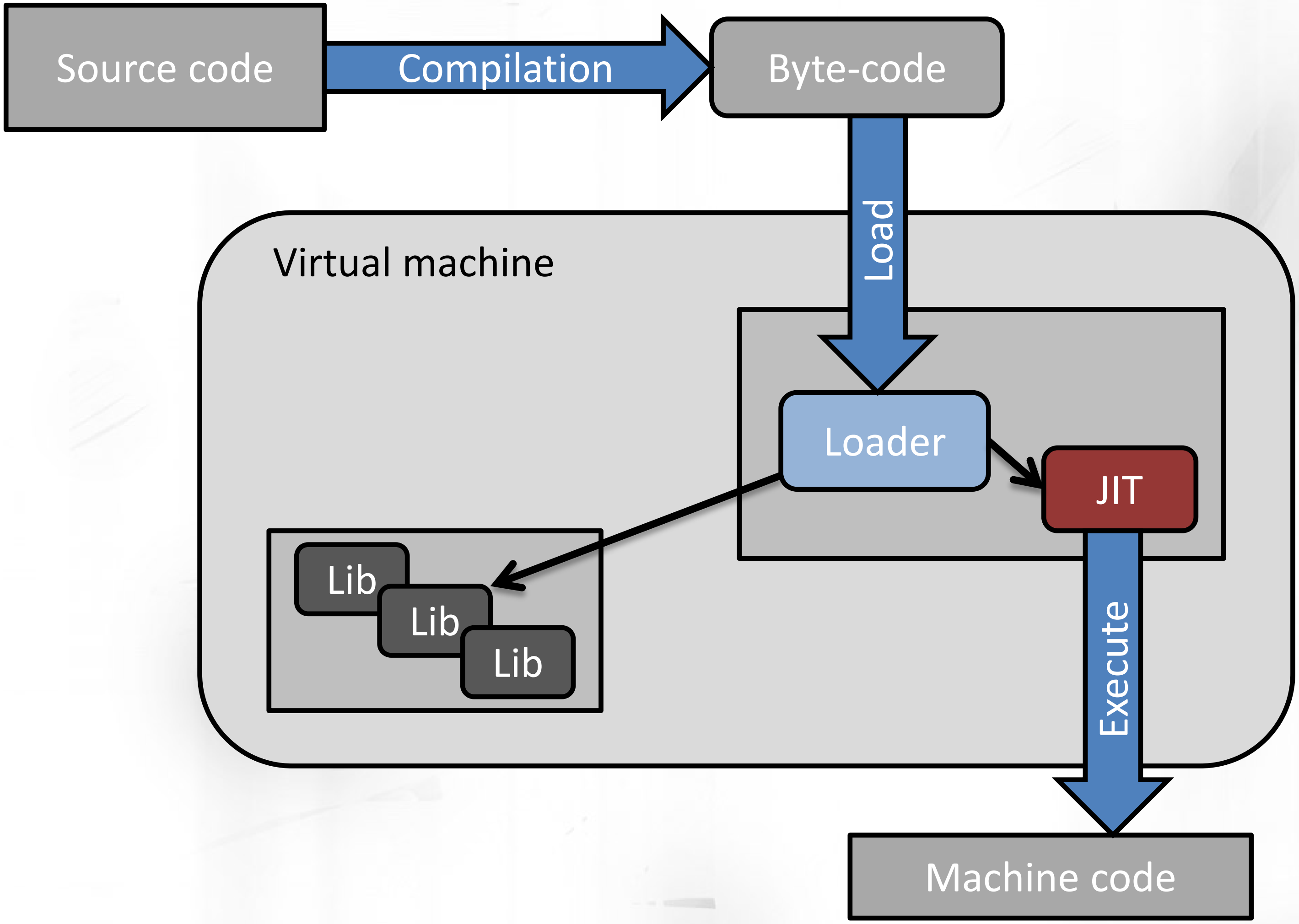
```
                                        /*
                                       +
                                      +
                                     +
                                     +
                                     [         >i>n[t
                                      */   #include<stdio.h>
                        /*2w0,1m2,]_<n+a m+o>r>i>=>(['0n1'0)1;
                     */int/**/main(int/**/n,char**m){FILE*p,*q;int        A,k,a,r,i/*
                   #uinndcelfu_dset<rsitcdti_oa.nhs>i/_*/;char*d="P%"   "d\n%d\40%d"/**/
                 "\n%d\n\00wb+",b[1024],y[]="yuriyurarararayuruyuri*daijiken**akkari~n**"
          "/y*u*k/riin<ty(uyr)g,aur,arr[a1r2a82*y2*/u*r{uyu}riOcyurhiyua**rrar+*arayra*="
       "yuruyurwiyuriyurara'rariayuruyuriyuriyu>rararayuruy9uriyu3riyurar_aBrMaPrOaWy^?"
      "*]/f]`;hvroai<dp/f*i*s/<ii(f)a{tpguat<cahfaurh(+uf)a;f}vivn+tf/g*`*w/jmaa+i`ni("/**
     */"i+k[>+b+i>++b++>l[rb";int/**/u;for(i=0;i<101;i++)y[i*2]^="~hktrvg~dmG*eoa+%squ#l2"
     ":(wn\"1l))v?wM353{/Y;lgcGp`vedllwudvOK`cct~[|ju {stkjalor(stwvne\"gt\"yogYURUYURI"[
     i]^y[i*2+1]^4;/*!*/p=(n>1&&(m[1][0]-'-'||m[1][1]  !='\0'))?fopen(m[1],y+298):stdin;
      /*y/riynrt~(^w^)],]c+h+a+r+**+*[n>)+{>f+o<r<(-m]    =<2<5<64;}-]-(m+;yry[rm*])/[*
       */q=(n<3||!(m[2][0]-'-'||m[2][1]))?stdout /*]{     }[*/:fopen(m[2],d+14);if(!p||/*
       "]<<*-]>y++>u>>+r >+u+++y>--u---r>++i+++"  <)<      ;[>-m-.>a-.-i.++n.>[(w)*/!q/**/)
    return+printf("Can "  "not\x20open\40%s\40"    ""      "for\40%sing\n",m[!p?1:2],!p?/*
  o=82]5<<+(+3+1+&.(+  m  +-+1.)<)<|<|.6>4>-+(>    m-      &-1.9-2-)-|-|.28>-w-?-m.:>([28+
 */"read":"writ");for  (   a=k=u= 0;y[u];  u=2    +u){y[k++   ]=y[u];}if((a=fread(b,1,1024/*
,mY/R*Y"R*/,p/*U*/)/*          R*/ )>/*U{  */   2&& b/*Y*/[0]/*U*/=='P' &&4==/*"y*r/y)r\}
*/sscanf(b,d,&k,& A,&          i,  &r)&&       !   (k-6&&k -5)&&r==255){u=A;if(n>3){/*
]&<1<6<?<m.-+1>3> +:+ .1>3+++     .   -m-)    -;.u+=++.1<0< <; f<o<r<(.;<([m(=)/8*/
u++;i++;}fprintf   (q,    d,k,          u      >>1,i>>1,r);u  = k-5?8:4;k=3;}else
  /*]>*/{(u)=/*{   p> >u  >t>-]s                >++(.yryr*/+(    n+14>17)?8/4:8*5/
     4;}for(r=i=0  ;  ;){u*=6;u+=                  (n>3?1:0);if    (y[u]&01)fputc(/*
      <g-e<t.c>h.a r  -(-).)8+<1.                   >;+i.(<)<     <)+{+i.f>([180*/1*
      (r),q);if(y[u   ]&16)k=A;if                                  (y[u]&2)k--;if(i/*
      ("^w^NAMORI; {   I*/==a/*"                                  )*/){/**/i=a=(u)*11
       &255;if(1&&0>=     (a=                                     fread(b,1,1024,p))&&
        ")]i>(w)-;} {                                             /i-f-(-m--M1-0.)<{"
         [ 8]==59/* */                                           )break;i=0;}r=b[i++]
            ;u+=(/**>>                                          *..</<<<)<[[;]**/+8&*
            (y+u))?(10-            r?4:2):(y[u]              &4)?(k?2:4):2;u=y[u/*
             49;7i\(w)/;}           y}ru\=*ri[             ,mc]o;n}trientuu ren (
             */]-(int)'`';}          fclose(                p);k= +fclose( q);
              /*] <*.na/m*o{ri{                           d;^w^;}  }^_^}}
               "    */   return  k-                 -1+   /*\'    '-`*/
                     (    -/*}/    */0x01          );     {;{     }}
                          ;           /*^w^*/            ;}
```

Byte code instrumentation

Byte code – intermediate representation between source code and machine code.

Java VM Dalvik VM AVM/AVM2 CLR …

Instrumentation byte-code (I)



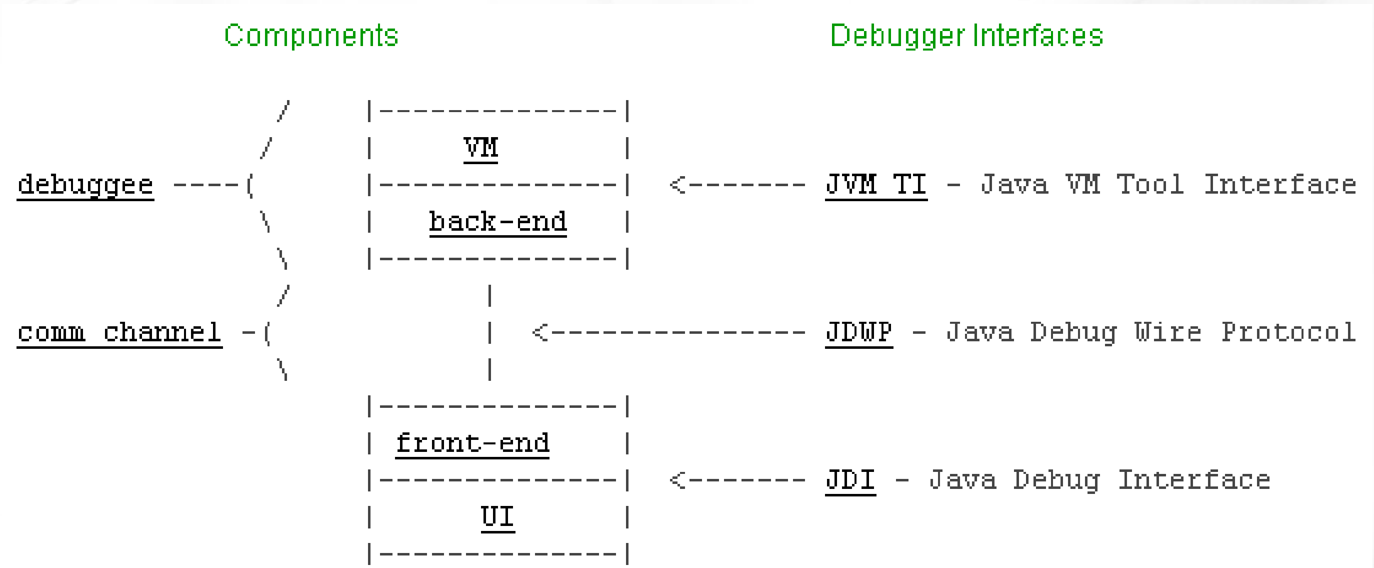
Instrumentation byte code (II)

- Byte-code
  - Static instrumentation
    - Static byte code instrumentation
  - Load-time instrumentation
    - Custom byte code loader
  - Dynamic instrumentation
    - Dynamic byte-code instrumentation

Instrumentation Java (I)

Mechanisms:

- java.lang.instrument package;
- Java Platform Debugger Architecture (JPDA) .



Instrumentation Java (II)

- Static instrumentation
  - Modification *.class files
- Load-time instrumentation
  - ClassFileLoadHook
  - Custom ClassLoader
- Dynamic instrumentation
  - ClassFileLoadHook -> RetransformClasses

Tools: Javassist, ObjectWeb ASM, BCEL, JOIE, reJ JavaSnoop, Serp, JMangler

Instrumentation .NET

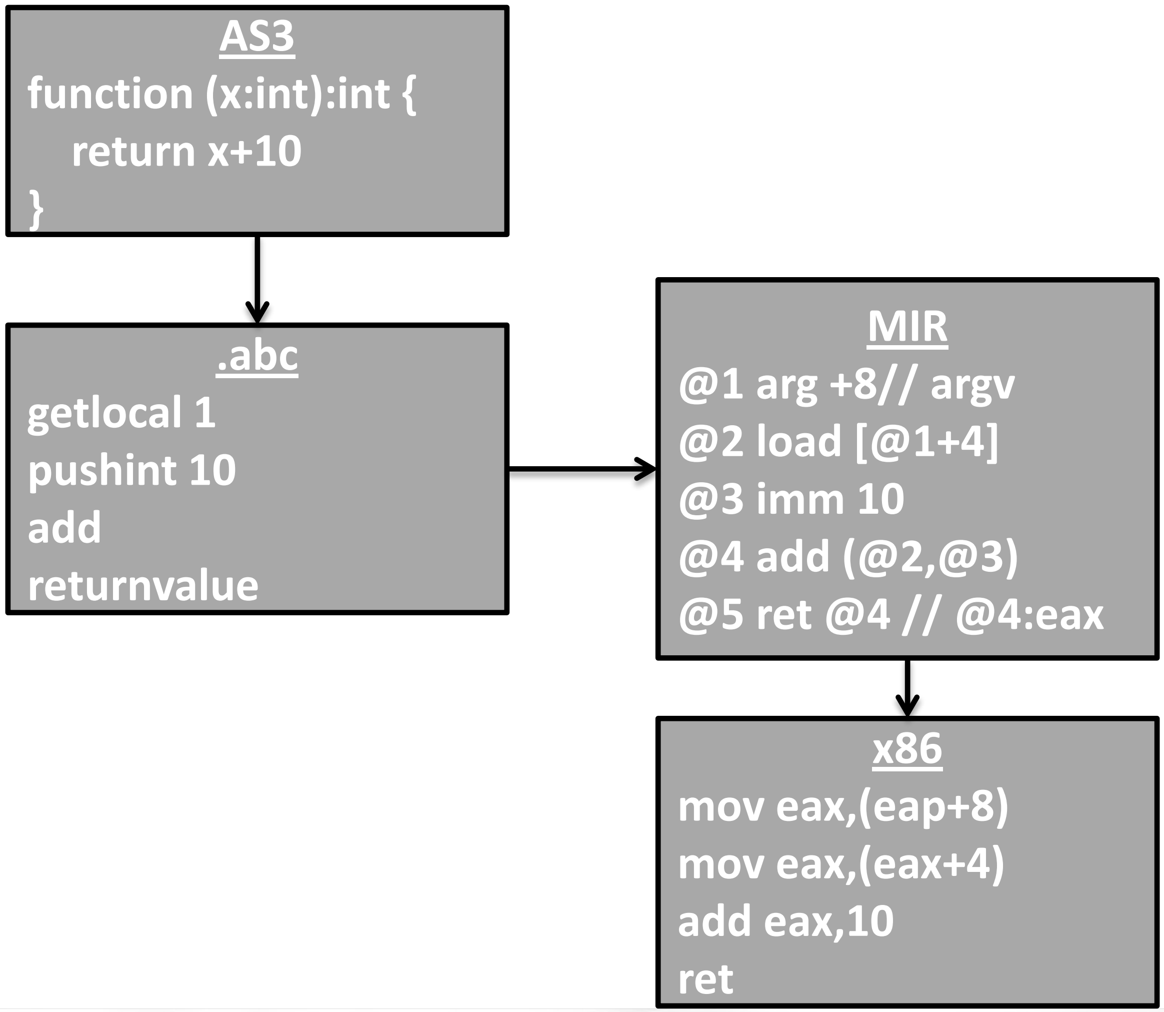
- Static instrumentation
  - Modification DLL files
- Load-time instrumentation
  - AppDomain.Load()/Assembly.Load()
  - Joint redirection
  - Via event handler

Tools: ReFrameworker, MBEL, RAIL, Cecil

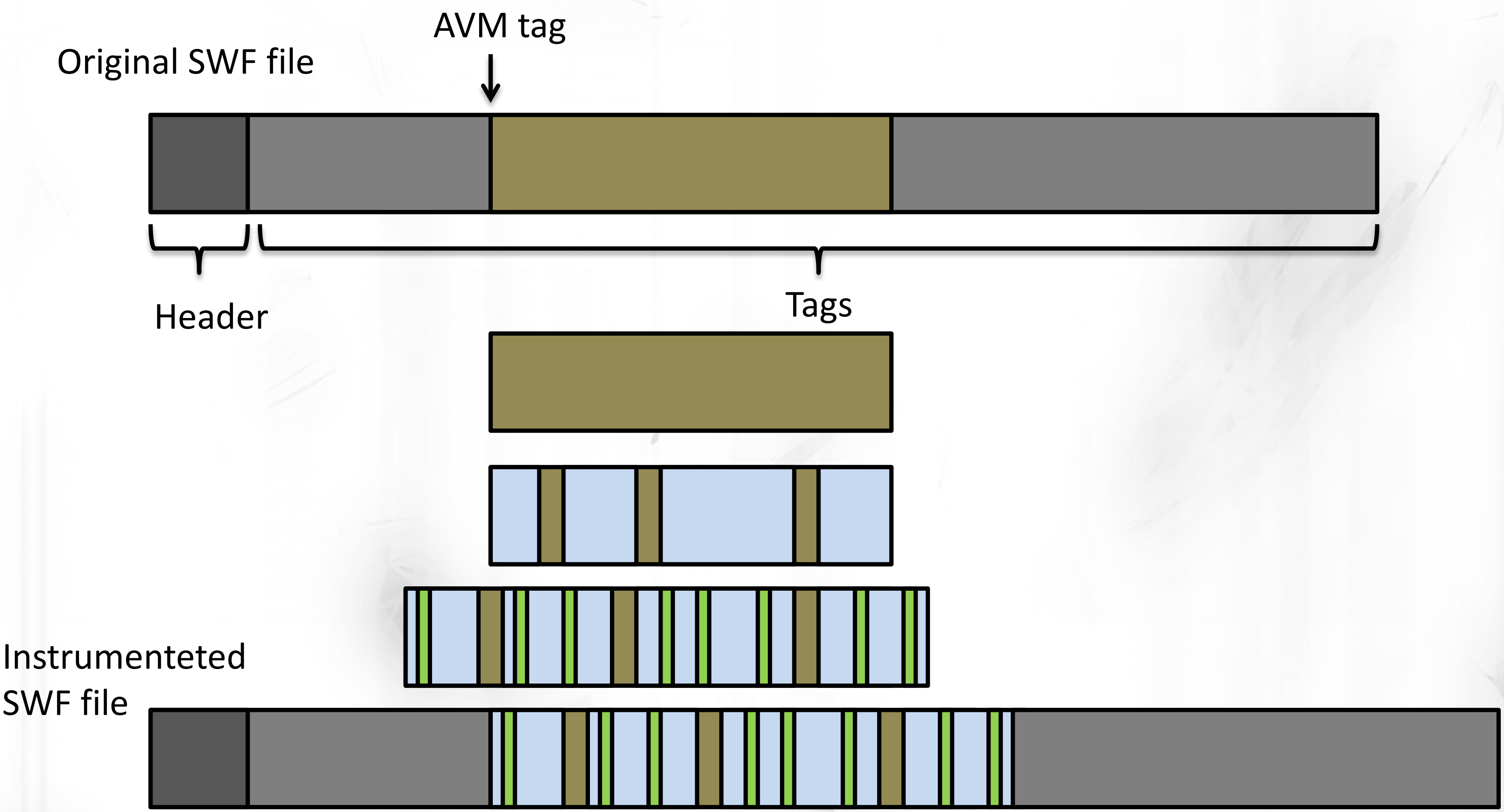
Instrumentation ActionScript (I)

- ActionScript2
  - AVM
  - Tags that (can) contain bytecode:
    - DefineButton (7), DefineButton2 (34), DefineSprite (39), DoAction (12), DoInitAction (59), PlaceObject2 (26), PlaceObject3 (70).
- ActionScript3
  - AVM2
  - Tags that (can) contain bytecode:
    - DoABC (82), RawABC (72).

AVM2 Architecture

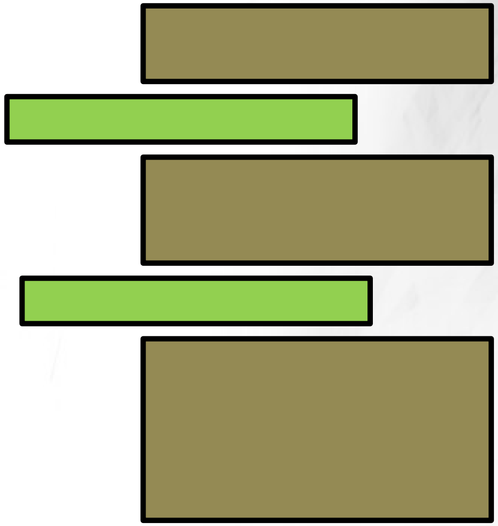


Instrumentation ActionScript (I)



Instrumentation AVM (II)

- Static instrumentation
  - Add :
    - trace()
    - dump()
    - debug()
    - debugfile()
    - debugline()
  - Modification:
    - Create own class + change class name = hook!



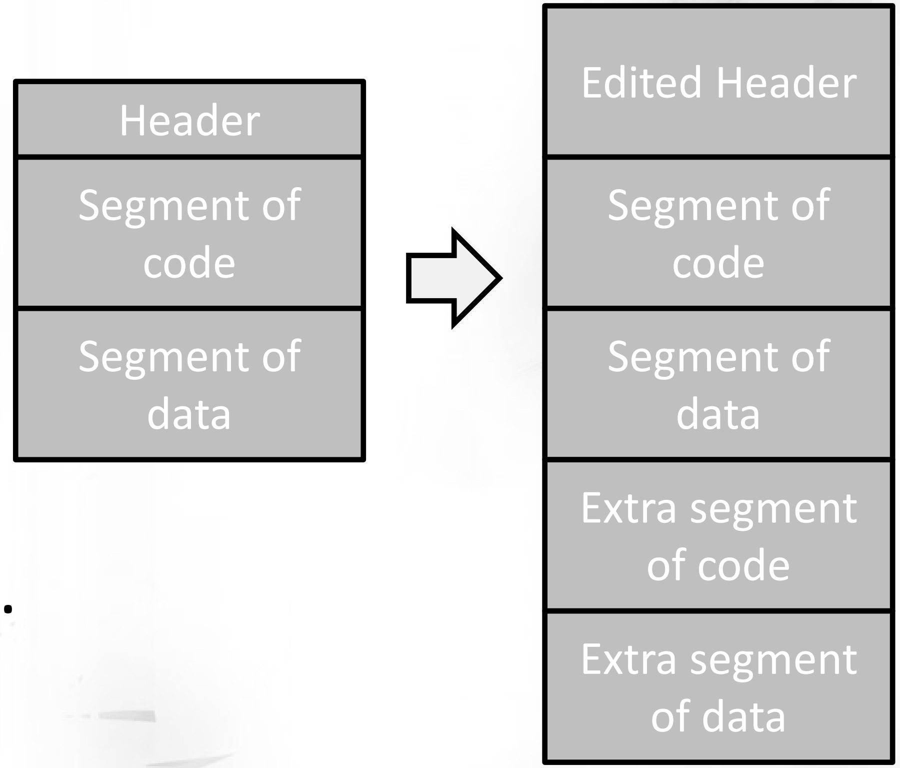
Instrumentation binary code

- The executable file
  - Static code instrumentation
    - Static binary instrumentation
- Process
  - Debuggers
    - Debugging API
  - Modifying call table/other structure
    - IAT
    - …
  - Dynamic code instrumentation
    - Dynamic binary instrumentation
- Hardware
  - Hardware debug features
    - Debug registers
    - Hardware debuggers
    - …
- Environment
  - Modifying call table
    - IDT, CPU MSRs, GDT, SSDT, IRP тable
    - …
  - Modifying OS options
    - SHIM
    - LD_PRELOAD
    - AppInt_DLLs
    - DLL injection
    - …
  - Reproduction of the environment
    - Emulation
    - Virtualization

Static Binary Instrumentation (I)

Static binary instrumentation/Physical code integration/Static binary code rewriting

- Realization:
  - With reallocation:
    - Level of segment;
    - Level of function;
  - Without reallocation.



Static Binary Instrumentation (II)

Reallocation:

1) Function Displacement + Entry Point Linking;

2) Branch Conversion;

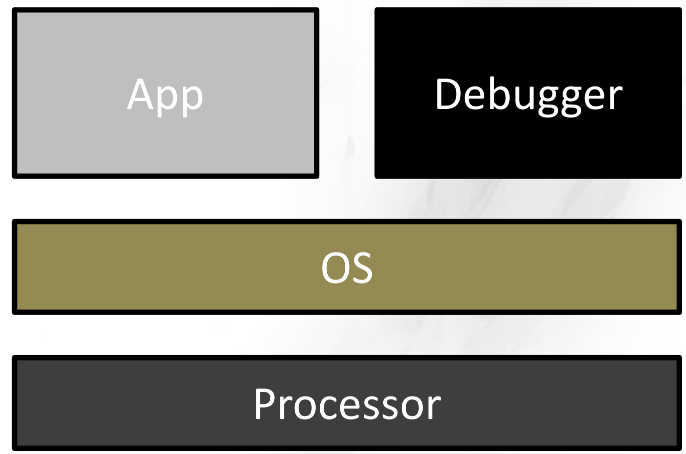
3) Instruction Padding;

4) Instrumentation.

Tools: DynInst, EEL, ATOM, PEBIL, ERESI, TAU, Vulcan, BIRD, Aslan(4514N)

Debuggers

- Breakpoints:
  - Software
  - Hardware



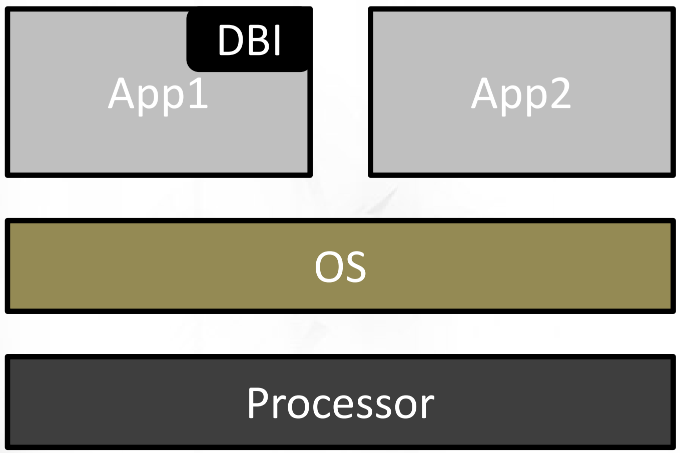
- Debugger + scripting:
  - WinDBG + pykd
  - OllyDBG + python = Immunity Debuggers
  - GDB + PythonGDB

- Python library's*: Buggery, IDAPython, ImmLIB, lldb, PyDBG, PyDbgEng, pygdb , python-ptrace , vtrace, WinAppDbg, …

*See "Python Arsenal for Reverse Engineering"

Dynamic Binary Instrumentation

Dynamic binary instrumentation/Virtual code integration/Dynamic binary rewriting



Tools: PIN, DynamoRIO, DynInst, Valgrind, BAP, KEDR, Fit, ERESI, Detour, Vulcan, SpiderPig

Dynamic Binary Instrumentation

- Dynamic Binary Instrumentation (DBI) is a process control and analysis technique that involves injecting instrumentation code into a running process.
- Dynamic binary analysis (DBA) tools such as profilers and checkers help programmers create better software.
- Dynamic binary instrumentation (DBI) frameworks make it easy to build new DBA tools.

- DBA tools consist:
  - instrumentation routines;
  - analysis routines.

Kinds of DBI

Mode:

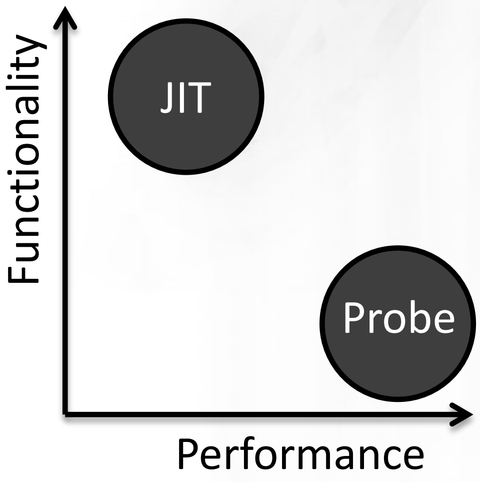
- user-mode;
- kernel-mode.

Mode of work:

- Start to finish;
- Attach.

Modes of execution:

- Interpretation-mode;
- Probe-mode;
- JIT-mode.



DBI Frameworks*

Frameworks	OS	Arch	Modes	Features
PIN	Linux, Windows, MacOS	x86, x86-64, Itanium, ARM	JIT, Probe	Attach mode
DynamoRIO	Linux, Windows	x86, x86-64	JIT, Probe	Runtime optimization
DynInst	Linux, FreeBSD, Windows	x86, x86-64, ppc32, ARM, ppc64	Probe	Static & Dynamic binary instrumentation
Valgrind	Linux, MacOS	x86, x86-64, ppc32, ARM, ppc64	JIT	IR – VEX, Heavyweight DBA tools

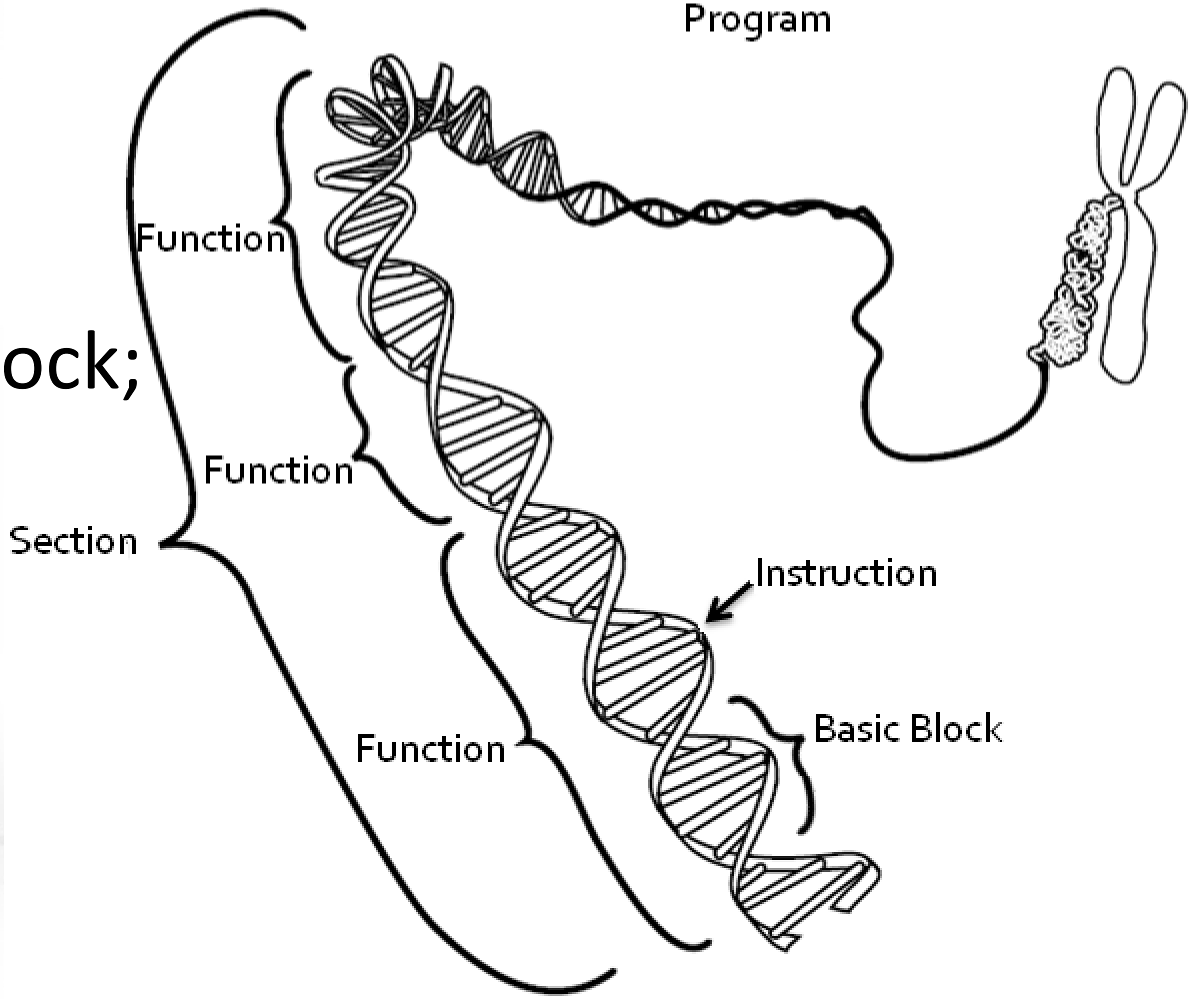
*For more details see "*DBI*:*Intro*" presentation from ZeroNights conference

Start work with DBI

```
C:\>type dbi.txt
//Launching PIN (JIT mode)
pin <pinargs> -t <pintool> <pintoolargs> -- <app> <appargs>
//Launching DynamoRIO (JIT mode)
drrun <drrunargs> -client <client> <clientargs> <app> <appargs>
//Launching Valgrind (JIT mode)
valgrind <valgrindargs> --tool=<toolname> <app> <appargs>

//Launching PIN (Probe mode)
pin -probe -t <pintool> <pintoolargs> -- <app> <appargs>
//Launching DynamoRIO (Probe mode)
drrun -mode probe -client <client> <clientargs> <app> <appargs>

//Attaching to a process in PIN
pin <pinargs> -t <pintool> <pintoolargs> -pid <app_pid>

C:\>_
```

Levels of granularity

- Instruction;
- Basic Block*;
- Trace/Superblock;
- Function;
- Section;
- Events;
- Binary image.



Self-modifying code & DBI

```
void InsertSmcCheck() {
    traceAddr = (VOID *) TRACE_Address(trace);
    traceSize = TRACE_Size(trace);
    TraceCopyAddr = malloc(traceSize);
    if (traceCopyAddr != 0) {
        memcpy(TraceCopyAddr, traceAddr, traceSize);

    TRACE_InsertCall(trace, IPOINT_BEFORE, (AFUNPTR)DoSmcCheck, IAGR_PTR, traceAddr,
                    IAGR_PTR, traceCopyAddr, IAGR_UINT32, traceSize, IAGR_CONTEXT, IAGR_END);
    }
}
void DoSmcCheck(VOID * traceAddr, VOID * traceCopyAddr, USIZE traceSize, CONTEXT * ctxP) {
    if (memcmp(traceAddr, traceCopyAddr, traceSize) != 0) {
        smcCount++;
        free(traceCopyAddr);
        CODECACHE_InvalidateTrace((ADDRINT)traceAddr);
        PIN_ExecuteAt(ctxP);
    }
}
void main (int argc, char **argv) {
    PIN_Init(argc, argv);
    TRACE_AddInstrumentationFunction(InsertSmcCheck, 0);
    PIN_StartProgram();
}
```

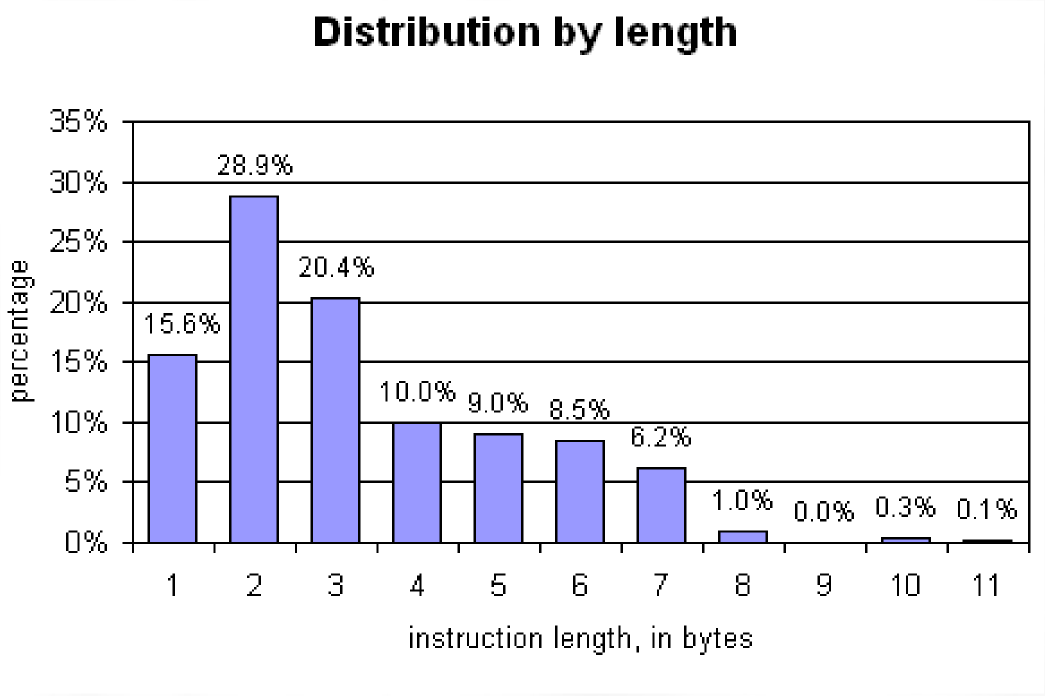
Overhead

$$O = X + Y$$
$$Y = N*Z$$
$$Z = K+L$$

O – Tool Overhead;
X – Instrumentation Routines Overhead;
Y – Analysis Routines Overhead;
N – Frequency of Calling Analysis Routine;
Z – Work Performed in the Analysis Routine;
K – Work Required to Transition to Analysis Routine;
L – Work Performed Inside the Analysis Routine.

Rewriting instructions

- Platforms:
  - with fixed-length instruction;
  - with variable-length instructions.



Rewriting code (I)

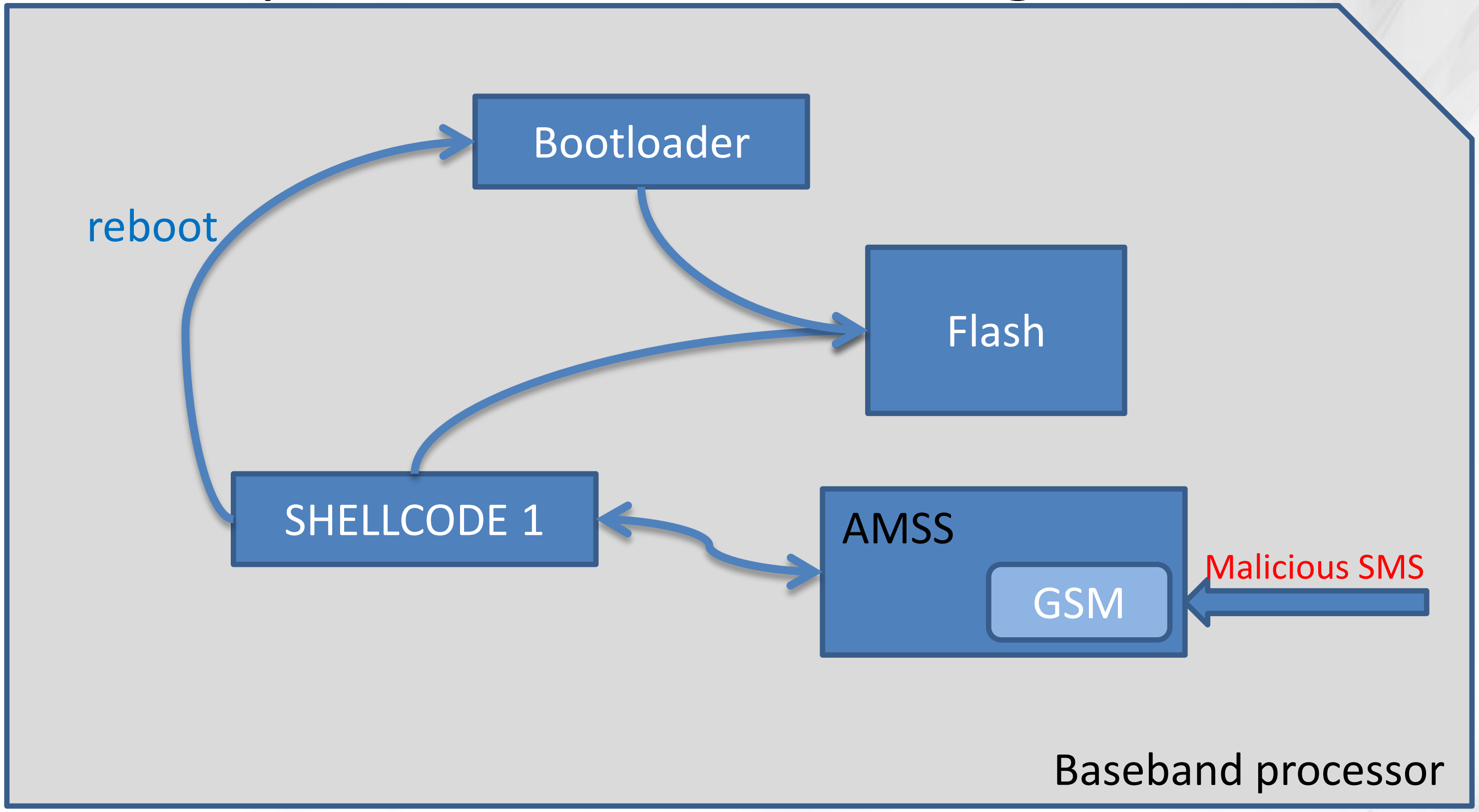
- Easy / simple / boring / regular example
  - Rewriting prolog function

```
; int __stdcall CFileOpenBrowser___GetPBItemFro
?_GetPBItemFromCSIDL@CFileOpenBrowser@@QAEHKPAU

csidl= dword ptr  8
psfi= dword ptr  0Ch
ppidl= dword ptr  10h

mov     edi, edi
push    ebp
mov     ebp, esp
push    ebx
push    esi
mov     esi, [ebp+ppidl]
push    esi              ; ppidl
push    [ebp+csidl]      ; csidl
xor     ebx, ebx
push    ebx              ; hwnd
call    ds:__imp__SHGetSpecialFolderLocation@12
test    eax, eax
jl      short loc_763AA4C3
```

```
                db 5 dup(90h)

; =============== S U B R O U T I N E ===============

; public: int __thiscall CFileOpenBrowser::OnSetCurso
?OnSetCursor@CFileOpenBrowser@@QAEHXZ proc near
                                        ; CODE XREF:
                cmp     dword ptr [ecx+0D4h], 0
                jz      short loc_763AA80F
                push    7F02h           ; lpCursorNa
                push    0               ; hInstance
```

Rewriting code (II)

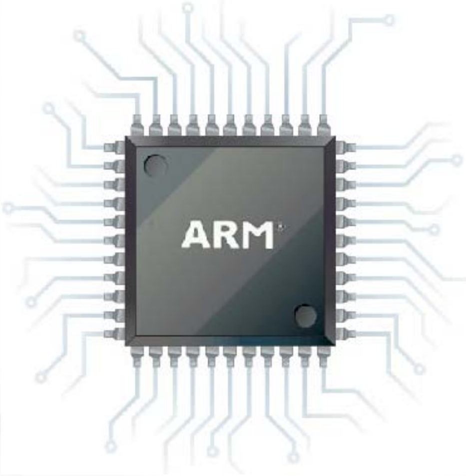
- Hardcore example:
  - Mobile phone firmware rewriting



Instrumentation in ARM

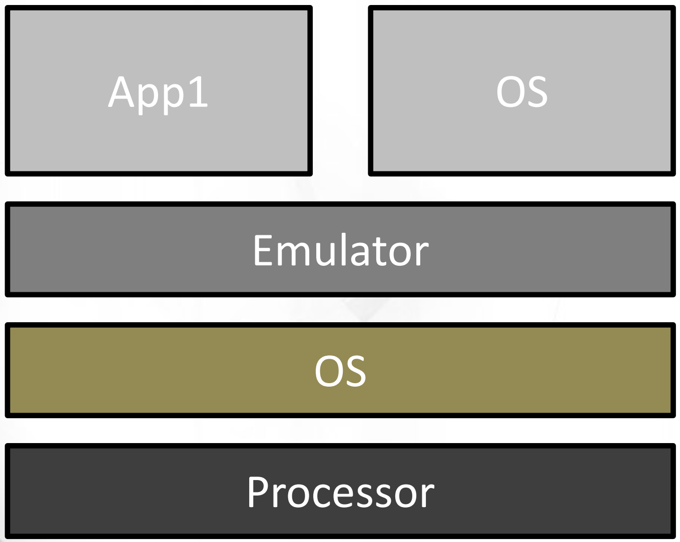
ARM modes:

- ARM
  - Length(instr) = 4 byte
- Thumb
  - Length(instr) = 2 byte
- Thumb2
  - Length(instr) = 2/4 byte
- Jazzle



For more detail see “A Dynamic Binary Instrumentation Engine for the ARM Architecture” presentation.

Emulation



Instrumentation & Bochs

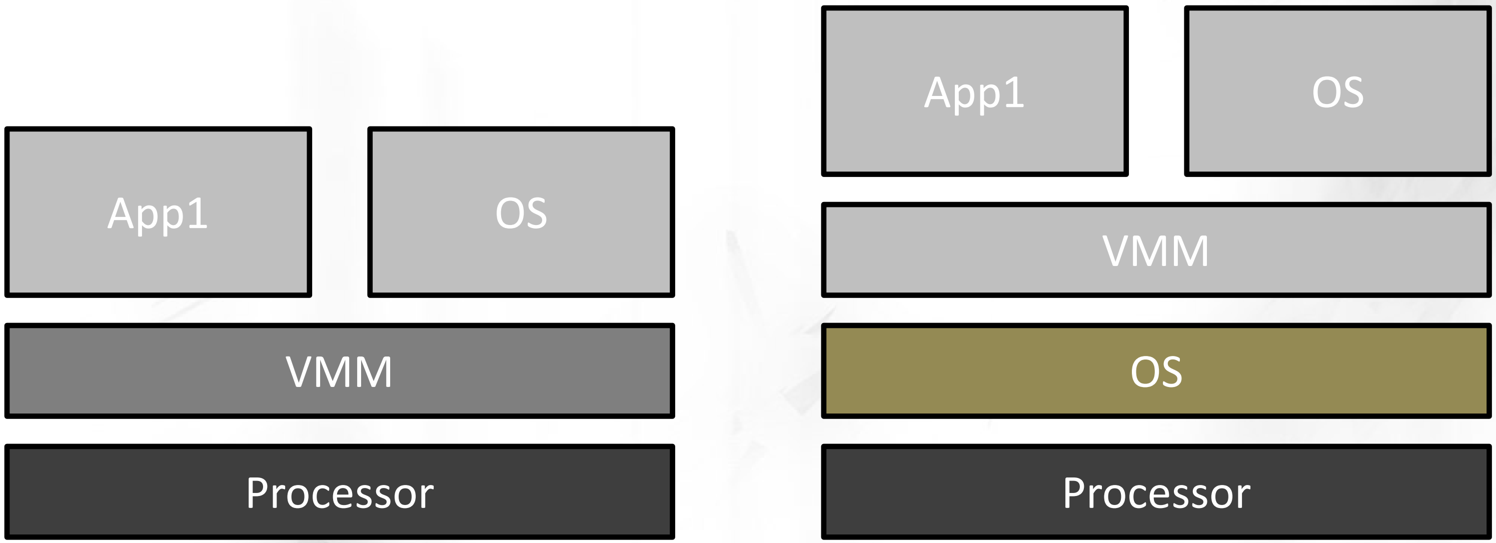
- Bochs can be called with instrumentation support.

```
./configure [...] --enable-instrumentation
./configure [...] --enable-instrumentation="instrument/stubs"
```

- C++ callbacks occur when certain events happen:
  - Poweron/Reset/Shutdown;
  - Branch Taken/Not Taken/Unconditional;
  - Opcode Decode (All relevant fields, lengths);
  - Interrupt /Exception;
  - Cache /TLB Flush/Prefetch;
  - Memory Read/Write.

- “bochs-python-instrumentation” patch by Ero Carrera

Virtualization



Native VMM

Hosted VMM

*VMM - Virtual Machine Monitor

Instrumentation & virtualization

Stages:

1. Save the VM-exit reason information in the VMCS;
2. Save guest context information;
3. Load the host-state area;
4. Transfer control to the hypervisor;
5. Run own code.

*VMCS - Virtual Machine Control Structure

Instrumentation in Mobile World

Mobile Platform	Language	Executable file format
Android	Java	Dex
iOS	Objective-C	Mach-O
Windows Phone	.NET	PE

Conclusion

One can implement instrumentation of everything!

Contact

Twitter: @evdokimovds

E-mail: d.evdokimov@dsecrg.com

Windows 8

- Apps:
  - C++ & DirectX
  - C# & XAML
  - HTML & JavaScript & CSS