

Descubriendo vulnerabilidades en ejecutables con SEA

Gustavo Grieco



SciPyCon Argentina 2013



Bugs y Exploits

Sea un programa **P** y una entrada **i**,

① Si $P(i)$ produce un resultado incorrecto o inesperado:

- ▶ **P** tiene un **bug**.
 - ▶ **i** es un **caso de prueba**.
-

② Si produce un resultado incorrecto o inesperado, y se puede sacar algún provecho

- ▶ **P** tiene una **vulnerabilidad**.
- ▶ **i** es un **exploit**.

Bugs y Exploits

¿Que sucede cuando alguien descubre bugs o vulnerabilidades?

1

2

3

Bugs y Exploits

¿Que sucede cuando alguien descubre bugs o vulnerabilidades?

- 1 Se **reporta** al fabricante.
- 2
- 3

Bugs y Exploits

¿Que sucede cuando alguien descubre bugs o vulnerabilidades?

- 1 Se **reporta** al fabricante.
- 2 El fabricante **soluciona** el bug.
- 3

Bugs y Exploits



¿Que sucede cuando alguien descubre bugs o vulnerabilidades?

- 1 Se **reporta** al fabricante.
- 2 El fabricante **soluciona** el bug.
- 3 **Todo el mundo** está contento.

Desgraciadamente..

Desgraciadamente..



Stuxnet

Desgraciadamente..



Stuxnet



Aurora

Desgraciadamente..



Stuxnet



Aurora



El dinero en el dinero..



- Google
 - ▶ Vulnerabilidad en **Chrome** o **Chromium**
- Pwn2own
 - ▶ Vulnerabilidad en **Adobe Acrobat Reader**
 - ▶ Vulnerabilidad en **Internet Explorer 10**

El dinero en el dinero..



- Google

- ▶ Vulnerabilidad en **Chrome** o **Chromium**

60.000 USD

- Pwn2own

- ▶ Vulnerabilidad en **Adobe Acrobat Reader**
 - ▶ Vulnerabilidad en **Internet Explorer 10**

El dinero en el dinero..



- Google

- ▶ Vulnerabilidad en **Chrome** o **Chromium**

60.000 USD

- Pwn2own

- ▶ Vulnerabilidad en **Adobe Acrobat Reader**
 - ▶ Vulnerabilidad en **Internet Explorer 10**

70.000 USD

El dinero en el dinero..



- Google

- ▶ Vulnerabilidad en **Chrome** o **Chromium**

60.000 USD

- Pwn2own

- ▶ Vulnerabilidad en **Adobe Acrobat Reader**
- ▶ Vulnerabilidad en **Internet Explorer 10**

70.000 USD
100.000 USD

Objetivos del proyecto

- Plataforma que permita analizar programas:
 - ▶ **Facilitar** la búsqueda vulnerabilidades concretas.
 - ▶ Definir **nuevos tipos** de vulnerabilidades.
 - ▶ Sin requerir el uso de **código fuente** del software analizado.
- Formar una comunidad y democratizar el acceso a la implementación de este tipo de herramientas.

¿Reinventando la rueda ..?



- !Exploitable (Microsoft Corp.)
- Automatic Generation of Control Flow Hijacking Exploits for Software Vulnerabilities (U. de Oxford)
- Unleashing MAYHEM on Binary Code (U. de Carnegie Melon)

¿Reinventando la rueda ..?



- !Exploitable (Microsoft Corp.)
- Automatic Generation of Control Flow Hijacking Exploits for Software Vulnerabilities (U. de Oxford)
 - “Unfortunately, I have **no idea** where that code is anymore...”
- Unleashing MAYHEM on Binary Code (U. de Carnegie Melon)

¿Reinventando la rueda ..?



- !Exploitable (Microsoft Corp.)
- Automatic Generation of Control Flow Hijacking Exploits for Software Vulnerabilities (U. de Oxford)
 - “Unfortunately, I have **no idea** where that code is anymore...”*
- Unleashing MAYHEM on Binary Code (U. de Carnegie Melon)
 - “We currently have **no plans** of releasing the source code of Mayhem”*

No es tan fácil..

```
movl 0x8(%EDX,%EBX,4), %EAX
```

Problema (1)

$EAX = mem[EDX + 4 * EBX + 8]$

Problema (2)

- ¿Y el desbordamiento?

$4 * EBX$

$EDX + 4 * EBX$

$EDX + 4 * EBX + 8$

No es tan fácil..

```
movl 0x8(%EDX,%EBX,4), %EAX
```

Problema (1)

$EAX = mem[EDX + 4 * EBX + 8]$

Problema (2)

- ¿Y el desbordamiento?

$4 * EBX$

$EDX + 4 * EBX$

$EDX + 4 * EBX + 8$

No es tan fácil..

```
movl 0x8(%EDX,%EBX,4), %EAX
```

Problema (1)

$EAX = mem[EDX + 4 * EBX + 8]$

Problema (2)

- ¿Y el desbordamiento?

- $4 * EBX$
- $EDX + 4 * EBX$
- $EDX + 4 * EBX + 8$

No es tan fácil..

```
movl 0x8(%EDX, %EBX, 4), %EAX
```

Problema (1)

$EAX = mem[EDX + 4 * EBX + 8]$

Problema (2)

- ¿Y el desbordamiento?

- ▶ $4 * EBX$
- ▶ $EDX + 4 * EBX$
- ▶ $EDX + 4 * EBX + 8$

No es tan fácil..

```
movl 0x8(%EDX, %EBX, 4), %EAX
```

Problema (1)

$EAX = mem[EDX + 4 * EBX + 8]$

Problema (2)

- ¿Y el desbordamiento?
 - ▶ $4 * EBX$
 - ▶ $EDX + 4 * EBX$
 - ▶ $EDX + 4 * EBX + 8$

No es tan fácil..

```
movl 0x8(%EDX, %EBX, 4), %EAX
```

Problema (1)

$EAX = mem[EDX + 4 * EBX + 8]$

Problema (2)

- ¿Y el desbordamiento?
 - ▶ $4 * EBX$
 - ▶ $EDX + 4 * EBX$
 - ▶ $EDX + 4 * EBX + 8$

Las herramientas apropiadas.



- 1 Representación abstracta de instrucciones ensamblador.
- 2 Teorías de satisfacibilidad (SMT).

Satisfiability Modulo Theories (SMT)

$$X = Y$$

- Lógica:
 - ▶ Lógica proposicional ($x \wedge y = z$)
 - ▶ Cuantificadores ($\forall x. x = y$)
- Matemática:
 - ▶ Operaciones lineales ($x + y = 1$)
 - ▶ Operaciones no lineales ($x * x = y$)
 - ▶ Funciones ($f(x) = y$)
- Números Binarios (longitud finita)
 - ▶ Aritmética de precisión finita ($x + y = 1$)
 - ▶ Operaciones de bits ($x \gg y = z$)
 - ▶ Arreglos ($x[y] = z$)

Satisfiability Modulo Theories (SMT)

$$X = Y$$

- Lógica:
 - ▶ ~~Lógica proposicional ($x \wedge y = z$)~~
 - ▶ ~~Cuantificadores ($\forall x. x = y$)~~
- Matemática:
 - ▶ ~~Operaciones lineales ($x + y = 1$)~~
 - ▶ ~~Operaciones no lineales ($x * x = y$)~~
 - ▶ ~~Funciones ($f(x) = y$)~~
- Números Binarios (longitud finita)
 - ▶ Aritmética de precisión finita ($x + y = 1$)
 - ▶ Operaciones de bits ($x \gg y = z$)
 - ▶ Arreglos ($x[y] = z$)

x86 / arm \Rightarrow Reverse
Engineering
Intermediate
Language

add
sub
mul
div
mod

x86 / arm \Rightarrow Reverse
Engineering
Intermediate
Language

x86 / arm \Rightarrow **R**everse
Engineering
Intermediate
Language

add
sub
mul
div
mod
or
and
xor
bsh
bisz

x86 / arm \Rightarrow Reverse
Engineering
Intermediate
Language

add
sub
mul
div
mod
or
and
xor
bsh
bisz
str
stm
ldm

x86 / arm \Rightarrow **R**everse
Engineering
Intermediate
Language

add
sub
mul
div
mod
or
and
xor
bsh
bisz
str
stm
ldm
jcc
nop
undef
unkn

x86 / arm \Rightarrow Reverse
Engineering
Intermediate
Language

add
sub
mul
div
mod
or
and
xor
bsh
bisz
str
stm
ldm
jcc
nop
undef
unkn

Symbolic Exploit Assistant (SEA)



Código
ensamblador
abstracto



01: $t_0 = \text{eax} + \text{ebx}$
02: $t_1 = \text{ebp} + \text{ecx}$
..



Conjunto de
restricciones

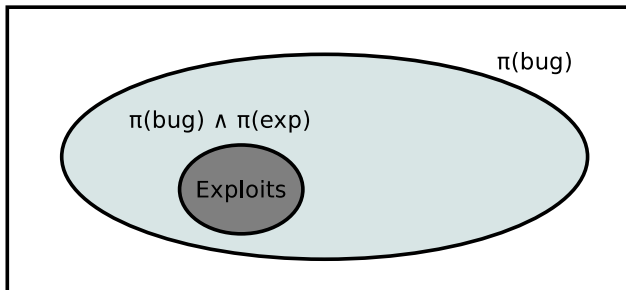
$\text{eax} = \text{ebx}$

$\text{ebx} = \text{ecx} + 1$

$\text{ecx} = \text{edx} \wedge \text{edx}$

¿Donde están los exploits?

Espacio de entradas de un programa vulnerable



¿Una vulnerabilidad?

```
01:  $t_0 = \text{eax} + \text{ebx}$   
02: if  $t_0 = 0$  then zf=1 else zf=0  
03: if zf  $\neq$  0 then goto 15  
15:  $t_1 = \text{ebp} + \text{ecx}$   
16:  $\text{mem}[t_1] := \text{eax}$   
..  
99: ret
```

Sobreescribiendo una dirección de memoria:

```
16:  $\text{mem}[\text{ebp} + 4] := 0\text{xdeadbeef} \Rightarrow t_1 = \text{ebp} + 4 \wedge \text{eax} = 0\text{xdeadbeef}$ 
```

Generando restricciones

```
01:  $t_0 = \text{eax} + \text{ebx}$   
02: if  $t_0 = 0$  then  $\text{zf} = 1$  else  $\text{zf} = 0$   
03: if  $\text{zf} \neq 0$  then goto 15  
15:  $t_1 = \text{ebp} + \text{ecx}$   
16:  $\text{mem}[t_1] := \text{eax}$   
..  
99: ret
```

Generando restricciones

```
01:  $t_0 = \text{eax} + \text{ebx}$ 
02: if  $t_0 = 0$  then  $\text{zf} = 1$  else  $\text{zf} = 0$ 
03: if  $\text{zf} \neq 0$  then goto 15
15:  $t_1 = \text{ebp} + \text{ecx}$ 
16:  $\text{mem}[t_1] := \text{eax}$ 
..
99: ret
```

$$\pi_{bug} = \{\text{zf} \neq 0, \text{ite}(t_0 = 0, \text{zf} = 1, \text{zf} = 0), t_0 = \text{eax} + \text{ebx}\}$$

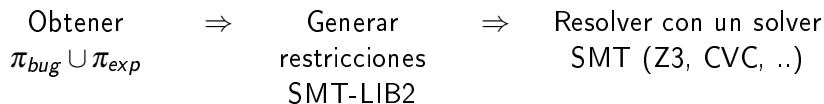
Generando restricciones

```
01:  $t_0 = \text{eax} + \text{ebx}$ 
02: if  $t_0 = 0$  then  $\text{zf} = 1$  else  $\text{zf} = 0$ 
03: if  $\text{zf} \neq 0$  then goto 15
15:  $t_1 = \text{ebp} + \text{ecx}$ 
16:  $\text{mem}[t_1] := \text{eax}$ 
..
99: ret
```

$$\pi_{bug} = \{\text{zf} \neq 0, \text{ite}(t_0 = 0, \text{zf} = 1, \text{zf} = 0), t_0 = \text{eax} + \text{ebx}\}$$

$$\pi_{exp} = \{\text{eax} = 0\text{xdeadbeef}, t_1 = \text{ebp} + 4, t_1 = \text{ebp} + \text{ecx}\}$$

Resolviendo restricciones



Resultado:

eax = 0xdeadbeef
ebx = 0x21524111
ecx = 0x00000004

¡SEA en acción!



(“Fork Me” in Github! → <https://github.com/neuromancer/SEA>)

Algunos desafíos..

- La selección de caminos es manual (por ahora)
- No está claro como propagar los tamaños de las entradas (escalabilidad, bla, bla).
- **Absoluta carencia** de tipos de datos algebraicos en Python (muy útil para para definir instruccioneso, operandos, etc).

¿Preguntas?

