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

Google

Aurora

The New York Times

¿?

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** — **70.000** USD
  - 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** — **70.000** USD
  - Vulnerabilidad en **Internet Explorer 10** — **100.000** USD

Objectivos 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 satisfabilidad (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 ($x[y] = z$)

REIL

REIL

x86 / arm $\Longrightarrow$ **R**everse
Engineering
Intermediate
Language

REIL

add
sub
mul
div
mod

x86 / arm $\Longrightarrow$ **R**everse
Engineering
Intermediate
Language

REIL

x86 / arm $\Longrightarrow$ **R**everse **E**ngineering **I**ntermediate **L**anguage

add
sub
mul
div
mod
or
and
xor
bsh
bisz

REIL

x86 / arm $\Longrightarrow$ **R**everse **E**ngineering **I**ntermediate **L**anguage

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

REIL

x86 / arm $\Longrightarrow$ **R**everse
Engineering
Intermediate
Language

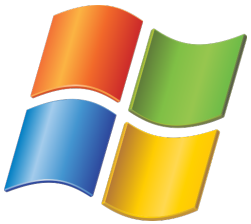
add
sub
mul
div
mod
or
and
xor
bsh
bisz
str
stm
ldm
jcc
nop
undef
unkn

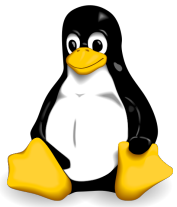
REIL

x86 / arm $\Longrightarrow$ **R**everse
Engineering
Intermediate
Language

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

Symbolic Exploit Assistant (SEA)





Mac

Código ensamblador abstracto

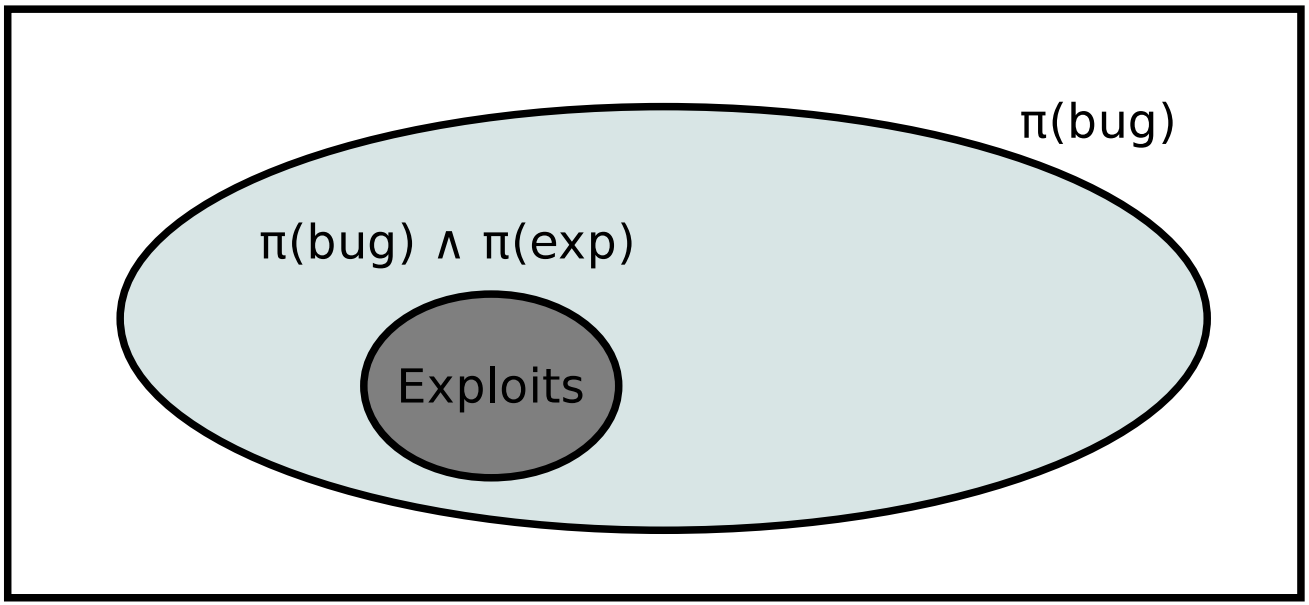
```
01: t0 = eax+ebx
02: t1 = ebp+ecx
..
```

Conjunto de restricciones

$$\begin{aligned} eax &= ebx \\ ebx &= ecx + 1 \\ ecx &= edx \wedge edx \end{aligned}$$

¿Donde están los exploits?

Espacio de entradas de un programa vulnerable



¿Una vulnerabilidad?

```
01: t0 = eax+ebx
02: if t0 = 0 then zf= 1 else zf= 0
03: if zf≠ 0 then goto 15
15: t1 = ebp+ecx
16: mem[t1] := eax
..
99: ret
```

Sobreescribiendo una dirección de memoria:

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

Generando restricciones

```
01: t0 = eax+ebx
02: if t0 = 0 then zf=1 else zf=0
03: if zf≠0 then goto 15
15: t1 = ebp+ecx
16: mem[t1] := eax
..
99: ret
```

Generando restricciones

```
01: t0 = eax+ebx
02: if t0 = 0 then zf = 1 else zf = 0
03: if zf ≠ 0 then goto 15
15: t1 = ebp+ecx
16: mem[t1] := eax
..
99: ret
```

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

Generando restricciones

```
01: t0 = eax+ebx
02: if t0 = 0 then zf= 1 else zf= 0
03: if zf≠ 0 then goto 15
15: t1 = ebp+ecx
16: mem[t1] := eax
..
99: ret
```

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

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

Resolviendo restricciones

Obtener $\pi_{bug} \cup \pi_{exp}$ $\Rightarrow$ Generar restricciones SMT-LIB2 $\Rightarrow$ Resolver con un solver SMT (Z3, CVC, ..)

Resultado:

$\text{eax} = 0xdeadbeef$

$\text{ebx} = 0x21524111$

$\text{ecx} = 0x00000004$

¡SEA en acción!

(“**Fork Me**” in Github! $\longrightarrow$ `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 algebráicos en Python (muy útil para para definir instruccioneso, operandos, etc).

¿Preguntas?