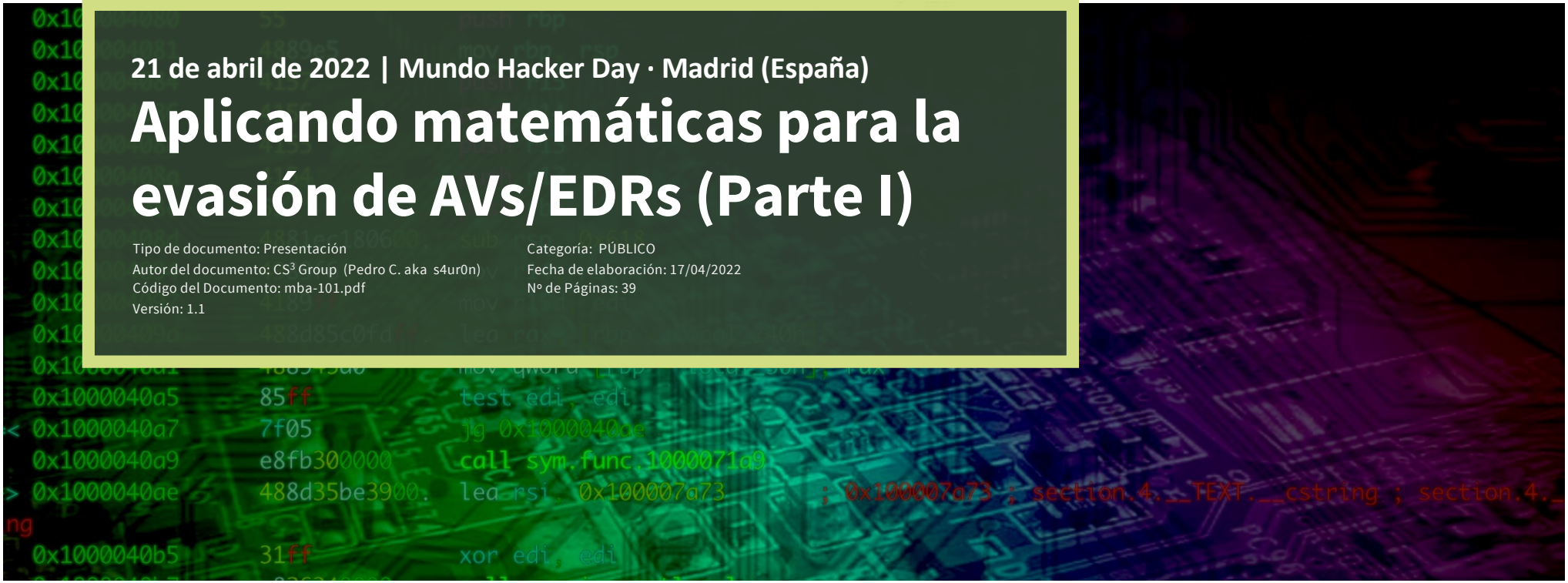
21 de abril de 2022 | Mundo Hacker Day · Madrid (España)

Aplicando matemáticas para la evasión de AVs/EDRs (Parte I)

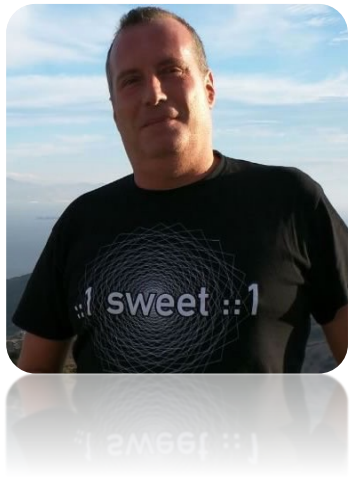
Tipo de documento: Presentación
Autor del documento: CS³ Group (Pedro C. aka s4ur0n)
Código del Documento: mba-101.pdf
Versión: 1.1

Categoría: PÚBLICO
Fecha de elaboración: 17/04/2022
Nº de Páginas: 39



Whoami

```
class PedroC:
    def __init__(self):
        self.name =      'Pedro Candel'
        self.email =     's4ur0n@s4ur0n.com'
        self.web =       'https://www.s4ur0n.com'
        self.nick =      '@NN2ed_s4ur0n'
        self.company =   'CS³ Group'
        self.webcorp =   'https://cs3group.com'
        self.role =      'Security Researcher'
        self.work =      [ 'Reversing', 'Malware', 'Offensive Security', '…' ]
        self.groups =    [ 'mlw.re', 'OWASP', 'NetXploit', '...' ]
```



mlw.re

OWASP
Open Web Application Security Project

CS³ Group

Formación en Seguridad
Cursos presenciales a medida impartidos en las instalaciones del cliente o las concertadas con prácticas reales desde el primer momento

Hacking Ético
Auditorías de caja negra, gris o blanca para aplicaciones web, sistemas y redes de comunicaciones

Análisis de Malware
Análisis de Malware automatizados y manuales con completos informes de comportamiento e indicadores de compromiso (IOC)

Ingeniería Inversa
Ingeniería Inversa para binarios de sistemas Windows de 32/64 bits, GNU/Linux de 32/64 bits, OSX Mach-O de 64 bits, ARM y firmwares

Exploiting
Desarrollo y adaptación de exploits para sistemas Windows de 32/64 bits, GNU/Linux de 32/64 bits, OSX Mach-O de 64 bits y Android

Desarrollo Seguro
Auditoría SAST, DAST, IAST y RASP para análisis de vulnerabilidades en el código de proyectos en Java, .Net, PHP, C/C++ y Cobol

Hardware Hacking
Análisis de vulnerabilidades en dispositivos hardware, sistemas embebidos y firmware con técnicas de ingeniería inversa

Seguridad en dispositivos móviles
Análisis estático, dinámico e instrumentación dinámica de aplicaciones Android (APK), iOS (IPA) y Windows Mobile (APPX)

Respuesta ante incidentes
Investigación remota de incidentes de seguridad, análisis de las situaciones y respuesta inmediata ante las amenazas

Forense
Adquisición y elaboración de informes periciales con garantía de imparcialidad y objetividad para todo tipo de sistemas de información

DevSecOps
Desarrollo, Seguridad y Operaciones en CSI (Continuous Security Integration) con pruebas automatizadas de seguridad para CI/CD

Intelligence
Recopilación, análisis y explotación de datos a gran escala con fuentes OSINT, SIGINT, HUMINT, Deep Web, redes P2P, etc.

SIGINT
Inteligencia de comunicaciones, análisis y auditoría de seguridad en señales y protocolos de radiofrecuencia (RF)

T.S.C.M.
Technical Surveillance Counter-Measures: Contramedidas electrónicas para detección y localización de dispositivos de escucha

Telecom
Análisis y auditoría GSM/3G/4G, implementación de servicios de operadores móviles virtuales (HLR, VLR, GGSN, Roaming voz y datos)

ATM
Análisis de vulnerabilidades, auditoría, forense, skimming, shimming y pruebas de blackbox para NCR, Hyosung, WRG, Diebold Nixdorf e Hitachi

PoS/TPV
Auditoría y cumplimiento de controles en terminales Verifone e Ingenico. Monitorización y transaccionabilidad completa según ISO 8583

LOPD/GPDR/Cumplimiento
LOPD, adaptación GPDR, ISO 27000, SGSI, análisis y gestión de riesgos, Políticas de seguridad, continuidad de negocio, ITIL, PCI DSS

1. Ofuscación

Code obfuscation through Mixed Boolean-Arithmetic (MBA) expressions

CS³ SECURITY SERVICES
https://cs3group.com

Code obfuscation through MBA expressions

La principal idea de la ofuscación del código es transformar un **programa P** de entrada en **otro equivalente P'** funcionalmente equivalente que es más complejo de analizarse y extraer su información (p.e. Reversing).

Code obfuscation through MBA expressions

Presenta muchos casos de uso:

- Protección del software (PI)
- Derechos digitales (DRM)
- Ocultación de “secretos”
- Evasión de firmas y patrones (AVs, YARAs, SIGMA…)
- Análisis de código complicado:
  - Esfuerzo
  - Tiempo
  - Dinero

Code obfuscation through MBA expressions

- **OFUSCACIÓN != CIFRADO**

  Las presentes técnicas, permiten **OFUSCAR** el código, en ningún caso pretenden ser un ***método de cifrado*** que es completamente distinto.

  Por lo tanto, la **DESOSFUCACIÓN**, puede dar lugar a la ***obtención del código original*** revelando secretos hardcodeados en nuestro código.

Code obfuscation through MBA expressions

- **MBA: Expresión aritmética mixta booleana**

  Es básicamente una expresión compuesta de operadores aritméticos (+/-/*…) y operadores booleanos (bitwise) (y/o/no/…)

  Por ejemplo:

$$E = (x \oplus y) + 2*(x \wedge y)$$

Code obfuscation through MBA expressions

Todo un completo tratado sobre MBA's y OC's puede encontrase en:

https://github.com/arnaugamez/talks/tree/master/2021/00_intent

> **@arnaugamez** (Arnau Gàmez), Hacker and **mathematician**. Founder, security researcher and trainer @FuraLabs. Senior malware reverse engineer for $vendor. Community @HackingLliure.

Code obfuscation through MBA expressions

A **polynomial MBA expression** consists of a **sum of terms**, each one composed by an n-**bit constant** a_i times the **product** of several **bitwise expressions** on **a number** t **of** n-**bit variables**.

$$E = \sum_{i \in I} a_i \cdot \left(\prod_{j \in J_i} e_{i,j}(x_1, \ldots, x_t) \right)$$

Code obfuscation through MBA expressions

$$E = \sum_{i \in I} a_i \cdot \left(\prod_{j \in J_i} e_{i,j}(x_1, \ldots, x_t) \right)$$

$$E = 43(x \wedge y \vee z)^2((x \oplus y) \wedge z \vee t) + 2x + 123(x \vee y)zt^2$$

Code obfuscation through MBA expressions

Ejemplo sencillo partiendo de la base:

$E_1 = x + y$

$E_2 = (x \oplus y) + 2*(x \wedge y)$

$E_3 = 151*(39*((x \oplus y) + 2 * (x \wedge y)) + 23) + 111$

Code obfuscation through MBA expressions

Podemos afirmar estas 2 premisas:

- Su **complejidad sintáctica** es **diferente**
- **Semánticamente** son **equivalentes**

Y **podemos demostrarlo**… por lo que sería igual en nuestro código emplear una expresión u otra.

Code obfuscation through MBA expressions

COMPLEJIDAD SINTÁCTICA del código generado (32 bits)

```
uint8_t E1(uint8_t x, uint8_t y)
{
  return x+y;
}
```

```
movzx edx, byte [var_4h]
movzx eax, byte [var_8h]
add   eax, edx
```

```
uint8_t E2(uint8_t x, uint8_t y)
{
  return (x^y)+2*(x&y);
}
```

```
movzx eax, byte [var_4h]
xor   al, byte [var_8h]
mov   edx, eax
movzx eax, byte [var_4h]
and   al, byte [var_8h]
add   eax, eax
add   eax, edx
```

```
uint8_t E3(uint8_t x, uint8_t y)
{
  return 151*(39*((x^y)+2*(x&y))+23)+111;
}
```

```
movzx eax, byte [var_4h]
xor   al, byte [var_8h]
movzx edx, al
movzx eax, byte [var_4h]
and   al, byte [var_8h]
movzx eax, al
add   eax, eax
add   eax, edx
imul  eax, eax, 0x27
add   eax, 0x17
mov   edx, 0xffffff97
imul  eax, edx
add   eax, 0x6f
```

Code obfuscation through MBA expressions

EQUIVALENCIA SE

Code obfuscation through MBA expressions

Linear MBA expr

Demostración d

We can easily verify t SMT solver like Z3.

```
from z3 import *
x = BitVec('x', 8)
y = BitVec('y', 8)
E = (x ^ y) + 2 * (x & y)    #
E_simp = x + y               #
prove (E == E_simp)          #
```

```
$ python eq.py
proved
```

2. Constantes Opacas

OC’s

Opaque Constants

El objetivo es “**ocultar**” una constante K.

Se combinan las expresiones MBA’s junto con **polinomios de permutación**.

Para cualquier polinomio de permutación “P”, existe otro “Q” que define su inversa. P.e. Para todos los valores de n-bits de X, se tiene que:

$$P(Q(x)) = x$$

Opaque Constants

Por ejemplo:

- **K** es la constante de n-bits a ocultar.
- **P** y **Q** son polinomios que cumplen con coeficientes de n-bits y actuando como inversa de 1 a 1. P(Q(x)) = x para todo x.
- **E** puede ser una expresión MBA con variables de n-bits no trivialmente igual a cero. P.e. $E(x_1, x_2, \ldots, x_t) = 0$ para cualquier variable $x_1, \ldots, x_t$

Opaque Constants

Luego:

- La constante **K** puede ser reemplazada por **P(E+Q(K))** para cualquier valor de n-bits de $x_1, \ldots, x_t$

Valores de 8 bits = 1 byte.

- $P(x) = 97x + 248x^2$
- $Q(x) = 161x + 136x^2$
- $E(x, y) = x - y + 2(\neg x \wedge y) - (x \oplus y)$

Opaque Constants

Creación del uint 8 c

```
from z3 import *

X = BitVec('X', 8)
def P(X): return 97*X + 248*X*X
def Q(X): return 161*X + 136*X*X

x = BitVec('x', 8)
y = BitVec('y', 8)
def E(x, y): return x-y + 2*(~x&y) -

K = BitVecVal(123, 8)

# Opaque Constant
OC = P(E(x,y) + Q(K))

# Apply basic simplification rules
print (simplify(OC))
```

```
te_oc.py

 +

5*y + 2*~(x | ~y) + 255*(x ^ y))*
+ 8*y + 240*~(x | ~y) + 8*(x ^ y))
```

Opaque Constants

Demostración que la [illegible] or uint_8 de **123**:

```
on check_oc.py
```

```
from z3 import *

x = BitVec('x', 8)
y = BitVec('y', 8)

def OC(x, y):
    return 195 + 97*x + 159*y +\
    194*~(x | ~y) + 159*(x ^ y) +\
    (163 + x + 255*y + 2*~(x | ~y)
    (232 + 248*x + 8*y + 240*~(x |

prove(OC(x, y) == 123)
```

3. Ejemplos

MBA’s y OC’s

CS³ SECURITY SERVICES
https://cs3group.com

Ejemplos

- **Vectores IV**
- **Dirección IPv4**
  - Decimal
  - Hexadecimal
  - Octal
- **Dirección IPv6**
- **byteArray (Shellcode)**
- **Argumentos de funciones**
- **Dirección de memoria**
  - 32 bits
  - 64 bits
- **Número de Syscall**
- Etc, etc, etc…

Ejemplos

Hemos visto una **única expresión** de MBA… pero ***¿existen más?***

- x+y = (x^y)+2*(x&y) = ...

Reformulando, podemos decir también que x+y es equivalente a…

- y-(~x)-1
- -(~x)-(~y)-2

Ejemplos

- 2*y-(~x)+(~y)
- (x|y)+y-(~x&y)
- y+(x&~y)+(x&y)
- (x|y)+(~x|y)-(~x)
- (x^y)+2*y-2*(~x&y)
- -(x^y)+2*y+2*(x&~y)
- -(x|~y)-(~x)+(x&y)-2
- 2*(x|y)-(~x&y)-(x&~y)
- (x^y)+2*(~x|y)-2*(~x)
- (~x&y)+(x&~y)+2*(x&y)

Ejemplos

- (x|~y)+(~x&y)-(~(x&y))+(x|y)
- 3*(x|~y)+(~x|y)-2*(~y)-2*(~(x^y))
- 2*(~(x^y))+3*(~x&y)+3*(x&~y)-2*(~(x&y))
- …

Ejemplos

Y otras expresiones como **x-y**, **x^y**, **x&y**, **x|y**, ...

- **x-y** : (x^y)+2*(x|~y) + 2
- **x^y** : (x|~y)+(~x|y)-2*(~(x|y))-2*(x&y)
- **x&y** : -~(x&y)+(~x|y)+(x&~y)
- **x|y** : (~x&y)+(x&~y)+(x&y)
- **2x-y** : 3*(x&~y)-(~x&y)+(~x|y)-~(x&y)
- **-2*y** : -(x&y)-2*(~x&y)+~(x|y)-~(x^y)
- ...

Ejemplos

¿Y con más variables? P.e. Con **3 variables** y **6-7 términos**

- 4*~(x|(y^z))-1*(y^(~x|(y&z)))-7*~x+4*~(x|(y|z))+8*(~x&(~y&z))+3*(~x&(y&z))
- -1*(z^~(~x&(y|z)))+3*(x|(y&z))+1*(z^(x&~y))+1*~(x|(y|z))-2*~(~x|(~y|z))-3*(~x&(y&z))-3*(x&(~y&z))
- 2*(~(x&~y)&~(x^(y^z)))+1*(~x&(~y|z))-3*((x&y)^(x^(y&z)))-3*~(x|(y|z))+3*~(~x|(y|z))-2*~(~x|(~y|z))-1*(~x&(~y&z))
- ...

Ejemplos

P.e. **3 variables** y **41 términos**

-1*(z^(x|~y))-
2*(z^~(~x&(y|z)))+9*((x&~y)|~(x^(y^z)))+8*(y^~(x|(y&z)))+1*~(x|z)-
7*(x|(~y|z))+7*~z-1*((x&z)^~(y&~z))-4*~(x&~y)+9*(z^(x&~y))-
9*((x&y)|~(y^z))+2*((y&~z)^(x|(y^z)))-3*((x&y)|~(y|~z))+4*(y^~(x&z))-
1*~(x^(y|z))-1*~(y|z)-2*(y^z)+7*(x^y)-3*(z|~(x|y))-
7*(y|(x&~z))+1*((x&z)^~(x^(y&z)))-4*(x|~z)+5*(y^(x|~z))-
2*(x^(~y&z))+7*(y^~(x&(y|z)))+6*((x&y)^(y|~z))-1*(~z&~(x^y))-
5*(x&(y^z))+6*(~(x&y)&~(x^(y^z)))-5*(y^(x|(~y&z)))-
9*((x&~y)|(x^(y^z)))+1*(~(x^y)&~(x^z))+1*(y^~(x&(y&z)))-7*~(x&(~y&z))-
15*~(x|(y|z))+14*~(x|(~y|z))+22*~(~x|(~y|z))+13*(~x&(~y&z))-
4*(~x&(y&z))+6*(x&(~y&z))+24*(x&(y&z))

Ejemplos

P.e. **4 variables** y **47 términos**

2*(((~(x|~y)|~(y^z))&~t)|(~(~x|(~y&z))&t))-9*(((x&y)&~t)|(((y&z)|(~x&(y|z)))&t))-
7*(((y^(x|(y&z)))&~t)|(((x&y)^(y|z))&t))-
1*((((y&~z)^(x|(y^z)))&~t)|((y|z)&t))+8*(((y^~(x&(y|z)))&~t)|((z^~(~x&(y&z)))&t))-
1*(((z^(x&(y|z)))&~t)|((z^(x|(~y|z)))&t))+9*(((x|~z)&~t)|(((x^y)&(x^z))&t))+8*(((~(x&y)&(x^(y^z)))&~t)|((y|(x^z))&t))+1
((((x&~y)|(y^z))&~t)|((~(x|y)|(x^(y^z)))&t))+3((((x|y)&~(y^z))&~t)|(((y&~z)^(x|(y^z)))&t))+1*(((z&(x|y))&~t)|((~(x^y)|(
x^z))&t))+1*((((x&z)|(y&~z))&~t)|((y^~(x|z))&t))+1*((((y&~z)^(x|(y^z)))&~t)|(((x^y)&~(x^z))&t))-
1*(((y^~(~x|(~y&z)))&~t)|(((x&y)^(y|z))&t))-
3*(((y&(x|z))&~t)|((y^(~x|(y^z)))&t))+6*(((~x&(~y|z))&~t)|((z^(x|y))&t))+5*(((z&~(x&y))&~t)|((~x|(y&z))&t))-
1*(((y^(x|(y|z)))&~t)|((y^(x&(~y|z)))&t))-1*((~(~x|(y&z))&~t)|((z^~(x&y))&t))-
7*((((x&z)^~(x^(y&z)))&~t)|((~y|(x^z))&t))-5*((~(x|~z)&~t)|(~(x^y)&t))+7*(((y^~(x&(y^z)))&~t)|((z&(x|~y))&t))-
9*(((z&~(x&~y))&~t)|(~(y|~z)&t))+2*(((z^(x|(~y&z)))&~t)|((y^~(x&z))&t))+4*((((x&z)^~(x^(~y&z)))&~t)|((x|z)&t))-
1*(((x^(y^z))&~t)|((y^(x|(y^z)))&t))+5*(((z^(x|(~y|z)))&~t)|((z^~(~x|(y&z)))&t))+1*((((x&z)|(y&~z))&~t)|((y^~(~x&(y&z)
))&t))+2*(((z|~(x|~y))&~t)|(((x|y)&~(x^(y^z)))&t))+7*(((z|(x&~y))&~t)|((y^~(x&~z))&t))+1*(((y&~(x&~z))&~t)|((z^(x|(y&
z)))&t))-26*~(x|(y|(z|t)))-11*~(x|(~y|(z|t)))-35*~(~x|(y|(z|t)))-12*~(~x|(~y|(z|t)))-14*~(x|(y|(~z|t)))-14*~(x|(~y|(~z|t)))-
3*~(~x|(y|(~z|t)))-15*~(~x|(~y|(~z|t)))-9*(~x&(~y&(~z&t)))-7*(~x&(y&(~z&t)))-34*(x&(~y&(~z&t)))-30*(x&(y&(~z&t)))-
8*(~x&(~y&(z&t)))-17*(~x&(y&(z&t)))+13*(x&(~y&(z&t)))-18*(x&(y&(z&t)))

Ejemplos

- Llamadas a M al **segundo que corresponda** seg quipo.

Ejemplos

- Recordar que poo… bién las **OC's** y…

```
union Valor
{
        uint32_t dword;

        struct
        {
                uint32_t val
        };

        struct
        {
                uint16_t wor
                uint16_t wor
        };

        struct
        {
                uint8_t byte0;
                uint8_t byte1;
                uint8_t byte2;
                uint8_t byte3;
        };
};
```

Ejemplos

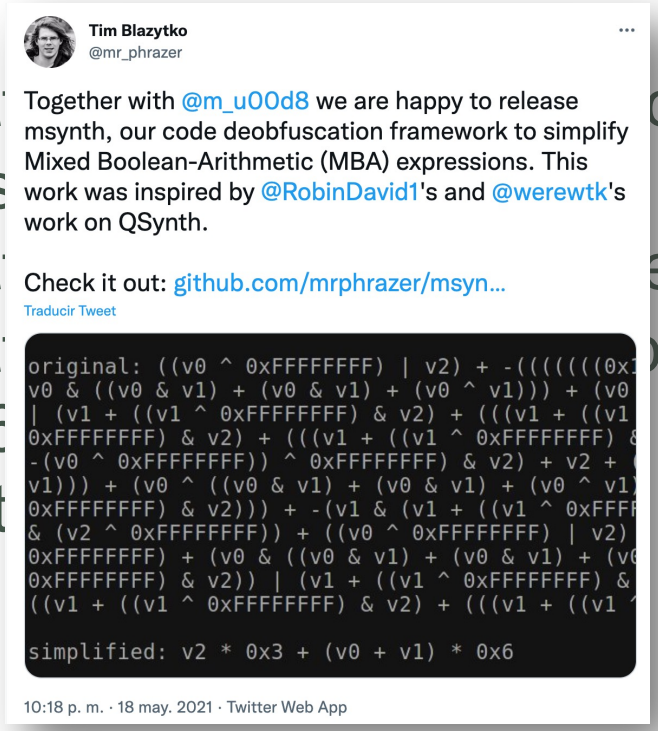
- Podemos combinarlo junto con **ofuscadores de código nativos**…

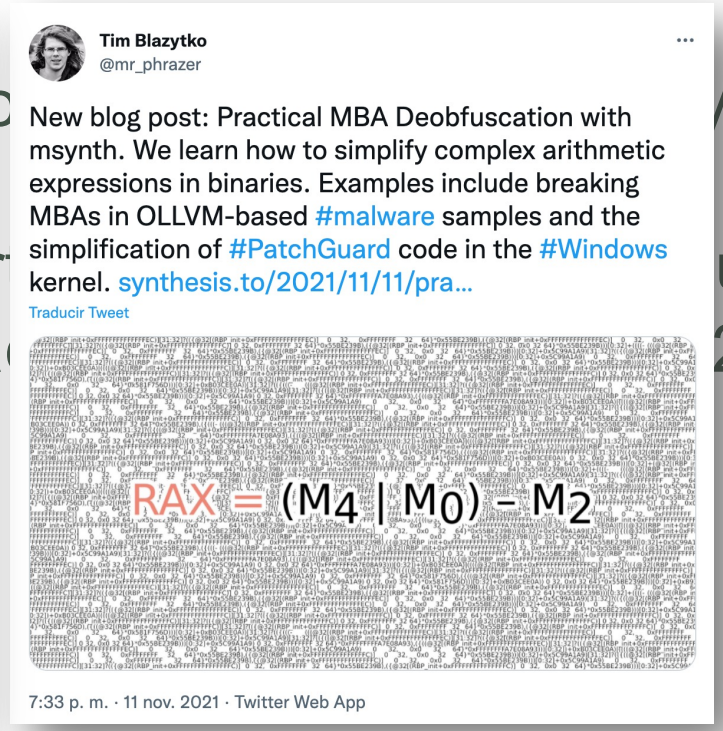
  - Obfuscator OLLVM 12.x (cadenas, sustituciones, bogus control-Flow, flatenning, etc…)
  - https://github.com/0x3f97/ollvm-12.x

Hasta que el “reverser” **nos maldiga** en su cueva…

Ejemplos

- No olvidemos que **existe el proceso inverso…**





4. Conclusiones

Conclusiones

- No ofuscar sólo el código y las cadenas.
- Ofuscar **todo tipo de operaciones susceptibles** (emplear 8/16/32/64 bits) según el destino y tipo.
- Recordar que - = +(-); * = +…n; …
- No emplear las optimizaciones de [illegible]ores (memoria, registros, etc.)

¿Preguntas?

¡Muchas gracias!