Deobfuscation and beyond

Vasily Bukasov

and

Dmitry Schelkunov

https://re-crypt.com

WWW.ZERONIGHTS.RU

Agenda

- We'll speak about obfuscation techniques which commercial (and not only) obfuscators use and how symbolic equation systems could help to deobfuscate such transformations
- We'll form the requirements for these systems
- We'll briefly skim over design of our mini-symbolic equation system and show the results of deobfuscation (and not only) using it

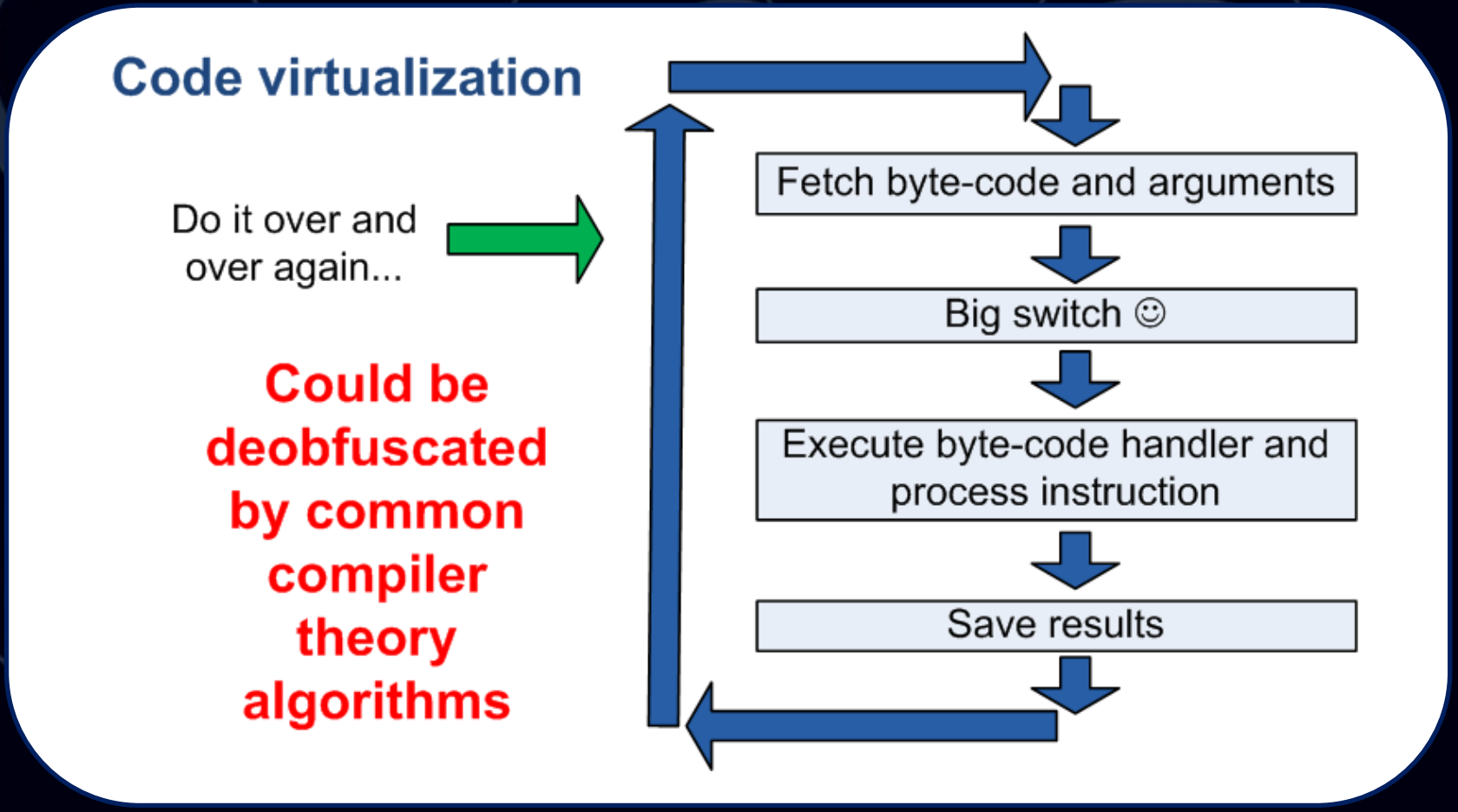
Software obfuscation

Is used for software protection against computer piracy

Is used for malware protection against signature-based and heuristic-based antiviruses

WWW.ZERONIGHTS.RU

Common obfuscation techniques



WWW.ZERONIGHTS.RU

Common obfuscation techniques

Recursive substitution

```
XOR EAX, EBX
```

```
PUSH EBX
NOT [ESP]
OR [ESP], EAX
NOT [ESP]
PUSH EBX
NOT [ESP]
AND [ESP], EAX
POP EAX
OR EAX, [ESP]
ADD ESP, 4
```

Could be deobfuscated by reverse recursive substitution

But … we prefer generic solutions ☺

WWW.ZERONIGHTS.RU

Common obfuscation techniques

Opaque predicates

```
PUSH EAX
PUSH EAX
ADD [ESP], EBX
PUSH EAX
AND [ESP], EBX
SHL [ESP], 1
POP EAX
SUB [ESP], EAX
XOR [ESP], EBX
POP EAX
XOR [ESP], EAX
POP EAX
JZ LABEL1
...
<DEAD CODE>
...
LABEL1:
...
```

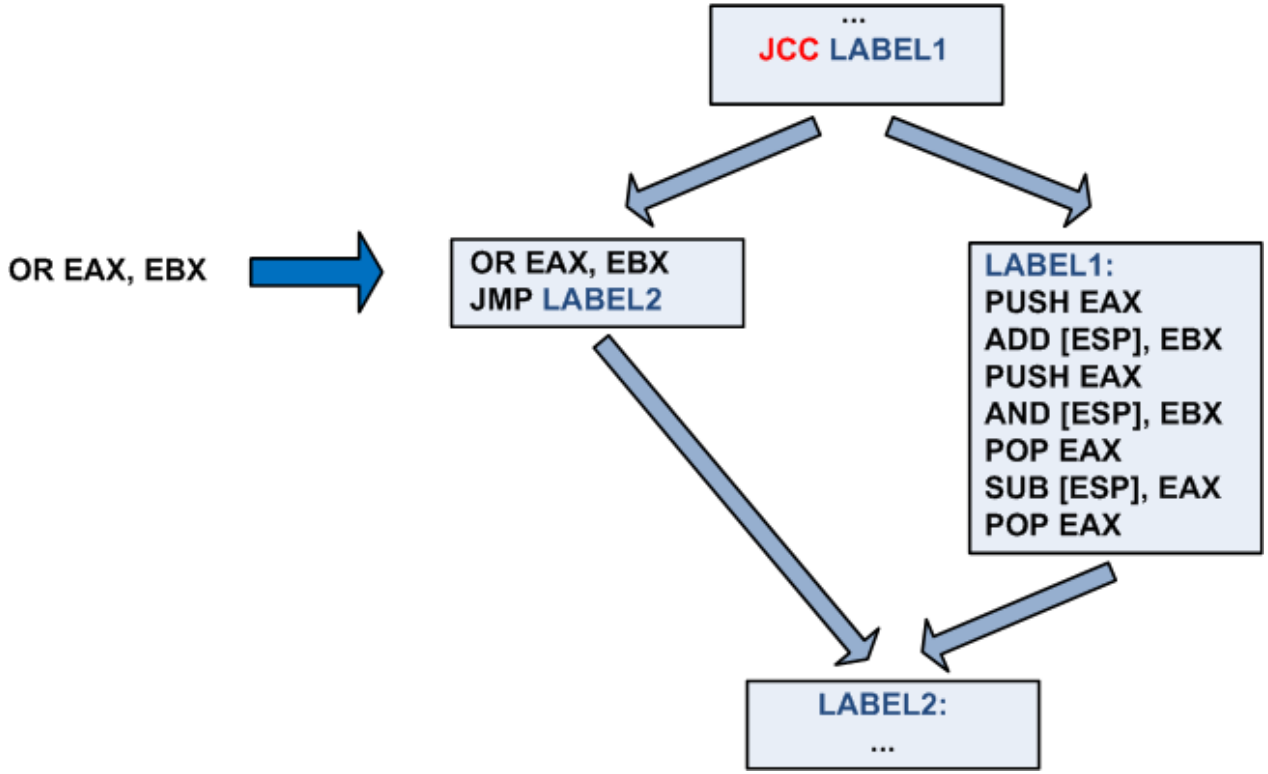
Garbage code

```
JCC LABEL1
;Meaningless block start
PUSH EBX
NOT [ESP]
AND [ESP], EAX
AND EAX, EBX
PUSH EAX
AND EAX, [ESP + 4]
XOR [ESP], EAX
POP EAX
XOR [ESP], EAX
POP EAX
XOR EAX, 0xFFFFFFFF
;Meaningless block end
LABEL1:
...
```

WWW.ZERONIGHTS.RU

Common obfuscation techniques

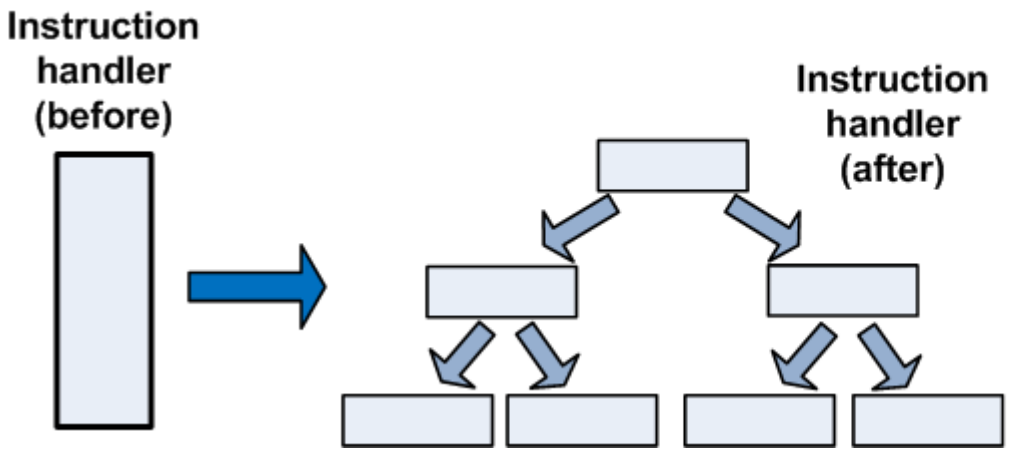
Code duplication



WWW.ZERONIGHTS.RU

Common obfuscation techniques

Code duplication in virtualization obfuscators



Very strong obfuscation technique but … not in this case ☺
We could choose a branch randomly and forget about the rest ☺

WWW.ZERONIGHTS.RU

Previous researches and products

- The Case for Semantics-Based Methods in Reverse Engineering, Rolf Rolles, RECON 2012

- Software deobfuscation methods: analysis and implementation, Sh.F. Kurmangaleev, K.Y. Dolgorukova, V.V. Savchenko, A.R. Nurmukhametov, H. A Matevosyan, V.P. Korchagin, Proceedings of the Institute for System Programming of RAS, volume 24, 2013

- CodeDoctor
  - deobfuscates simple expressions
  - plugin for OllyDbg and IDA Pro

WWW.ZERONIGHTS.RU

Previous researches and products

- VMSweeper
  - declares deobfuscation (devirtualization) of Code Virtualizer/CISC and VMProtect (works well on about 30% of virtualized samples)
  - not a generic tool (heavily relies on templates)
  - works as a decompiler not optimizer
  - weak symbolic equation system

- CodeUnvirtualizer
  - declares deobfuscation (devirtualization) of Code Virtualizer/CISC/RISC and Themida new VMs
  - not a generic tool (heavily relies on templates)
  - no symbolic equation system

WWW.ZERONIGHTS.RU

Previous researches and products

- Ariadne
  - complex toolset for deobfuscation and data flow analysis
  - includes a lot of optimization algorithms from compiler theory
  - no symbolic equation system
  - it seems to be dead ☹

- LLVM forks
  - are based on LLVM optimization algorithms (classical compiler theory algorithms)
  - we couldn't find any decently working version
  - are limited by LLVM architecture (How fast LLVM works with 500 000 IR instructions? How much system resources it requires?)

WWW.ZERONIGHTS.RU

The problem

Existing deobfuscation solutions are mostly based on classical compiler theory algorithms and too weak against modern obfuscators in the most of cases

WWW.ZERONIGHTS.RU

Solution

- Use symbolic equation system (SES) for deobfuscation
- Form input data for SES (translate source IR code to SES representation)
- Simplify expressions using SES
- Translate results from SES representation to IR
- Apply other deobfuscation transformations

WWW.ZERONIGHTS.RU

Symbolic equation system

Should work with mixed expressions

$0xff00 \& (0x100 + (0xff00 \& a)) = 0xff00 \& (0x100 + a)$

$0xff \& (a \& 0xff + b) = 0xff \& (a + b)$

$0xfffffffe \& a + 1 \& a = 0xffffffff \& a$

WWW.ZERONIGHTS.RU

Symbolic equation system

Should work with variables of various size

al = 1 => eax.1 = eax.0 & 0xffffff00 ^ 1

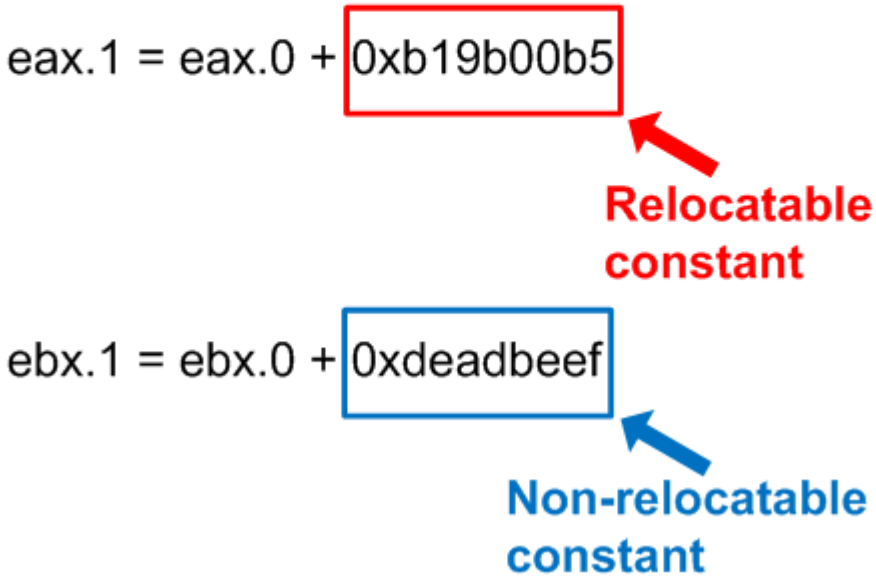
al = bl => eax.1 = eax.0 & 0xffffff00 ^ 0xff & ebx.0

ah = bl => eax.1 = eax.0 & 0xffff00ff ^ ((0xff & ebx.0) << 8)

WWW.ZERONIGHTS.RU

Symbolic equation system

Should preserve additional information about variables and constants



WWW.ZERONIGHTS.RU

Symbolic equation system

Should optimize expressions like following

eax.1 = (eax.0 ^ ebx.0) + ((eax.0 & ebx.0) << 1) =

eax.1 = (eax.0 | ebx.0) + (eax.0 & ebx.0) =

WWW.ZERONIGHTS.RU

Symbolic equation system

Should optimize expressions like following

eax.1 = (eax.0 ^ ebx.0) + ((eax.0 & ebx.0) << 1) = eax.0 + ebx.0

eax.1 = (eax.0 | ebx.0) + (eax.0 & ebx.0) = eax.0 + ebx.0

z3 returns something strange here ☺

WWW.ZERONIGHTS.RU

Symbolic equation system

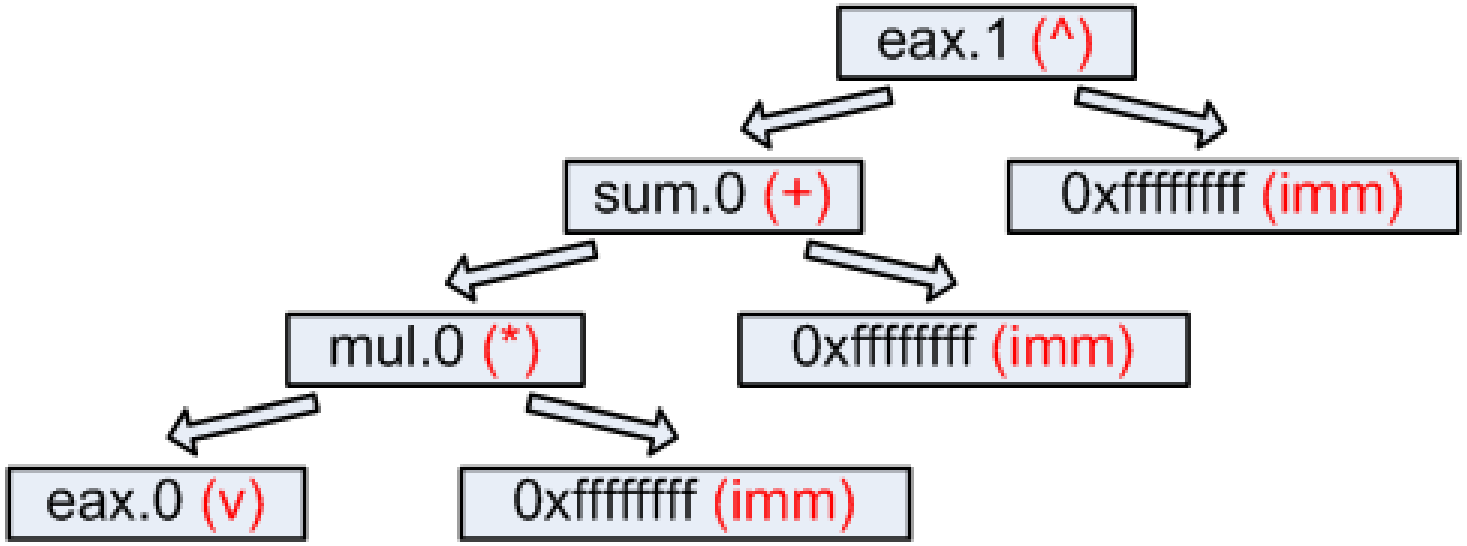
Unfortunately, we couldn’t find an appropriate third-party symbolic equation system engine and … we decided to create a new one for ourselves.

We called it Project Eq.

WWW.ZERONIGHTS.RU

Eq design

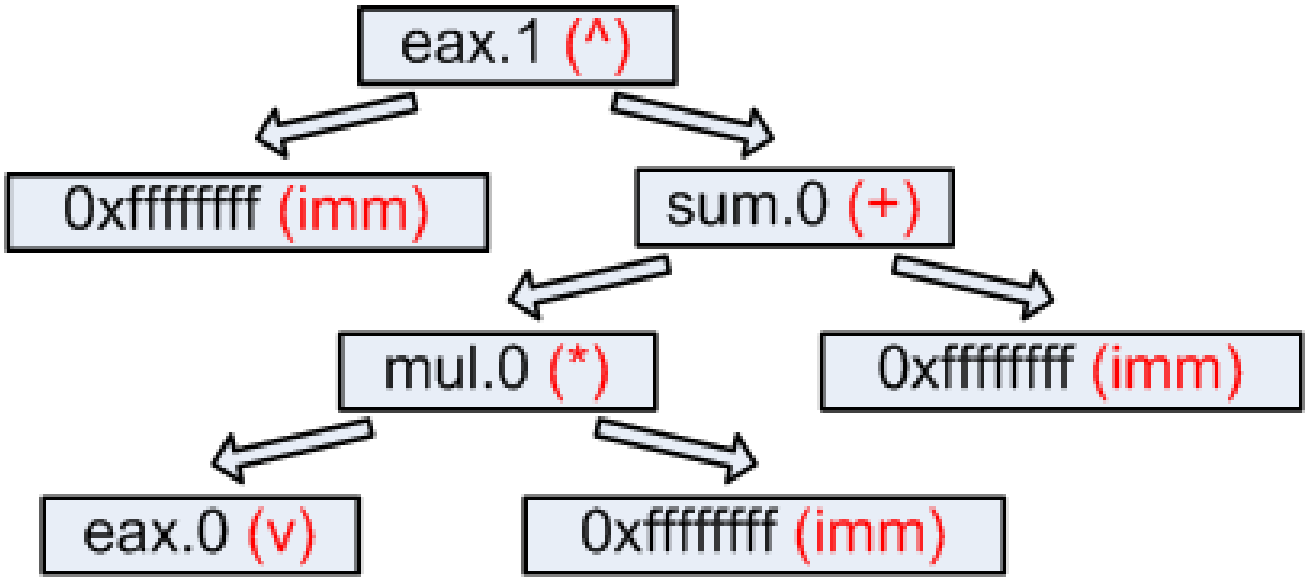
eax.1 = ((eax.0 * 0xffffffff) + 0xffffffff) ^ 0xffffffff



WWW.ZERONIGHTS.RU

Eq design

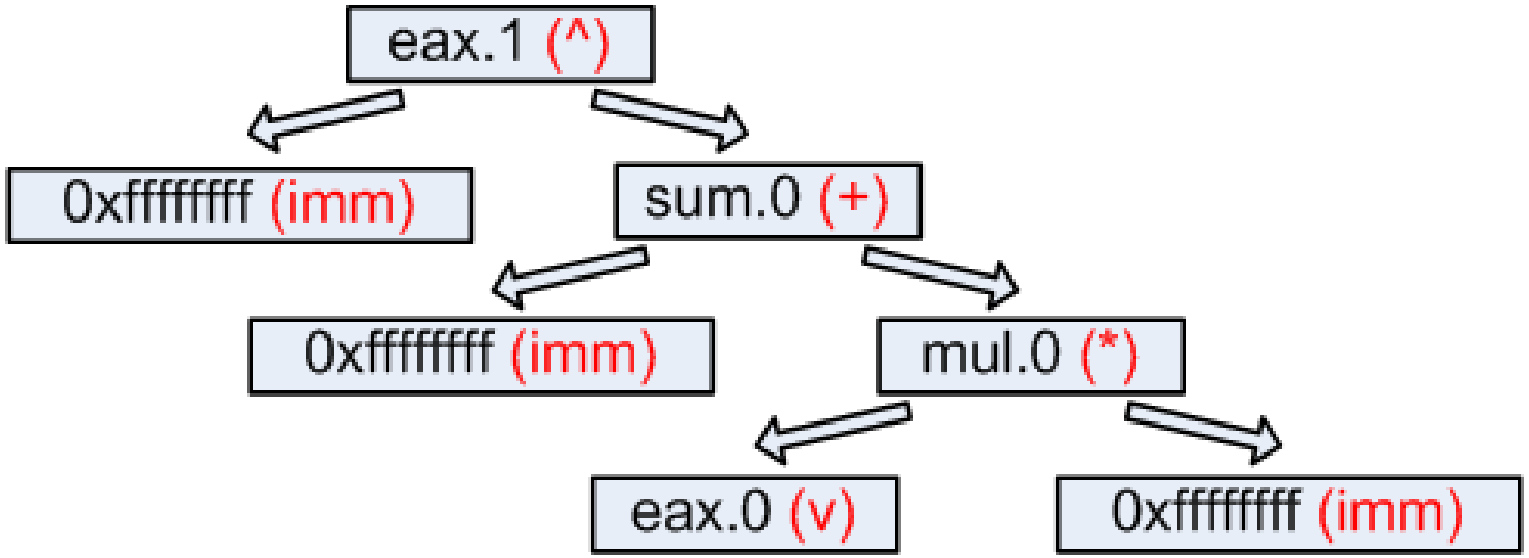
eax.1 = ((eax.0 * 0xffffffff) + 0xffffffff) ^ 0xffffffff



WWW.ZERONIGHTS.RU

Eq design

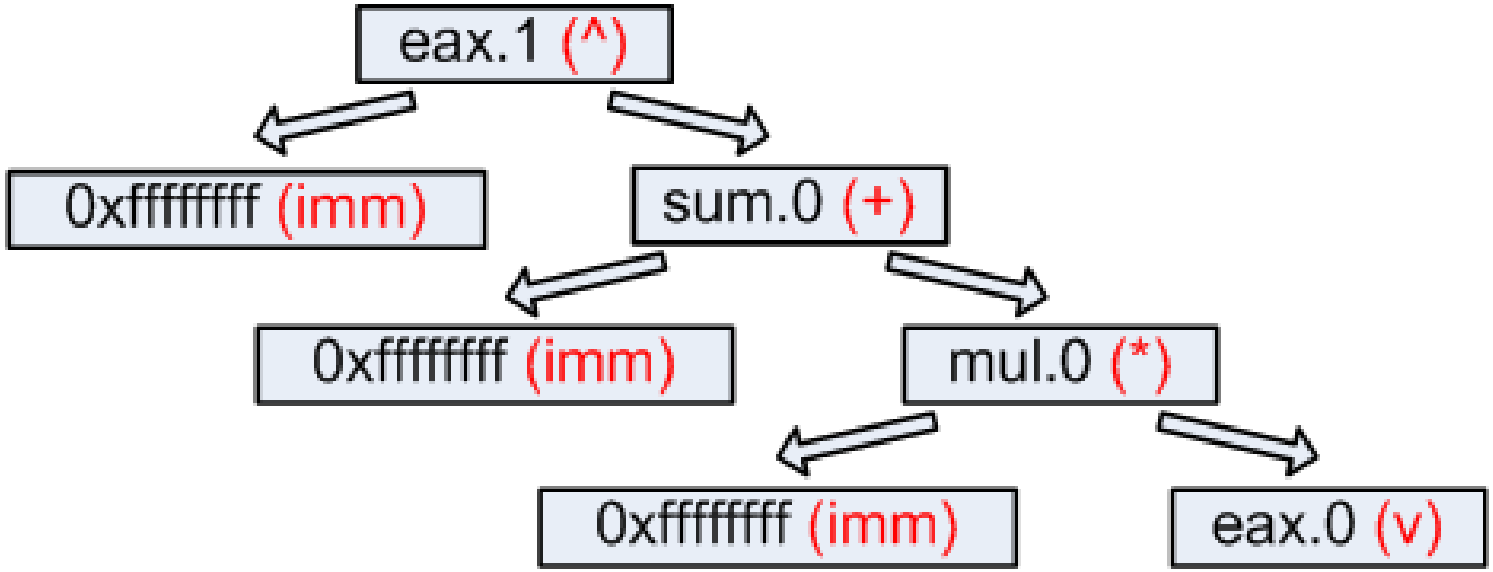
eax.1 = ((eax.0 * 0xffffffff) + 0xffffffff) ^ 0xffffffff



WWW.ZERONIGHTS.RU

Eq design

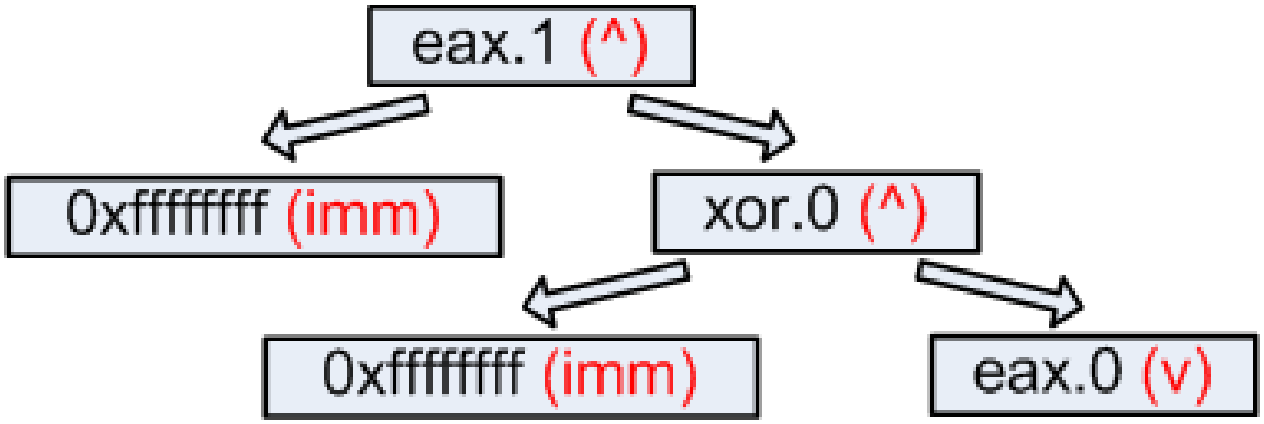
eax.1 = ((eax.0 * 0xffffffff) + 0xffffffff) ^ 0xffffffff



WWW.ZERONIGHTS.RU

Eq design

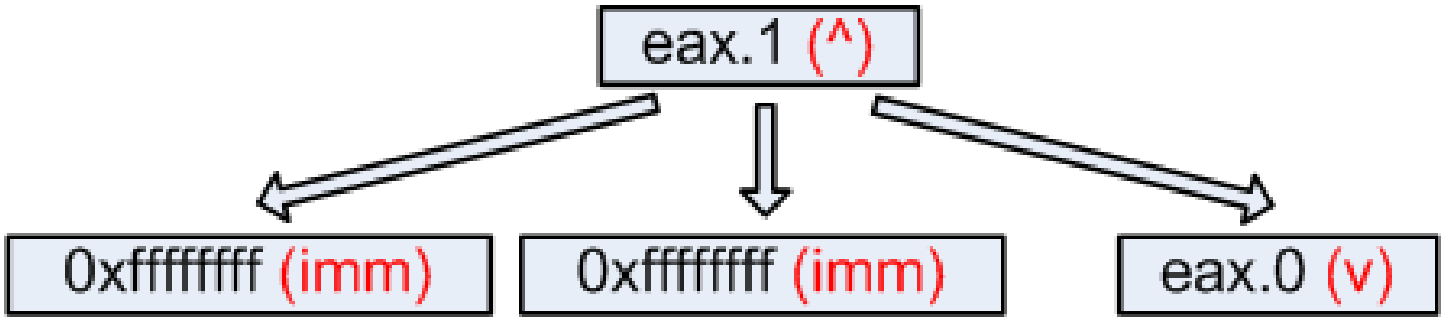
eax.1 = ((eax.0 * 0xffffffff) + 0xffffffff) ^ 0xffffffff



WWW.ZERONIGHTS.RU

Eq design

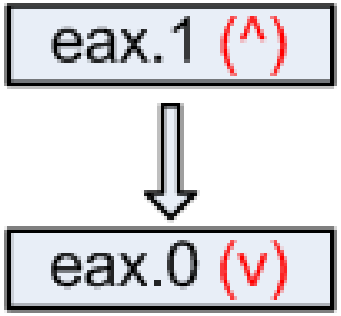
eax.1 = ((eax.0 * 0xffffffff) + 0xffffffff) ^ 0xffffffff



WWW.ZERONIGHTS.RU

Eq design

eax.1 = ((eax.0 * 0xffffffff) + 0xffffffff) ^ 0xffffffff



WWW.ZERONIGHTS.RU

Eq design

eax.1 = ((eax.0 * 0xffffffff) + 0xffffffff) ^ 0xffffffff

eax.0 (v)

eax.1 = eax.0

Profit! ☺

WWW.ZERONIGHTS.RU

Eq design

We are too lazy ☺
To represent boolean expressions we use Zhegalkin form:

eax | ebx = eax ^ ebx ^ eax & ebx

This “trick” allows us to use same optimization primitives for boolean and arithmetic expressions

WWW.ZERONIGHTS.RU

Eq in work

```
union rebx_type
{
	UINT32 rebx;
	WORD   rbx;
	BYTE   rblow[2];
};

void vmp_constant_playing(rebx_type &rebx)
{
	BYTE var0;
	union var1_type
	{
		UINT32 var;
		WORD   var_med;
		BYTE   var_low;
	} var1;

	var0 = rebx.rblow[0];
	rebx.rblow[0] = 0xe7;
	var1.var_med = rebx.rbx;
	var1.var_low = 0x18;
	rebx.rbx = var1.var_med;
	rebx.rblow[0] = var0;
}
```

A C++ sample of obfuscated code. It was borrowed ☺ from VMProtect

WWW.ZERONIGHTS.RU

Eq in work

Eq representation of the previous code (before optimization)

rebx.rebx.1 = ((0xff & rebx.rebx.0) ^ (0xffffff00 & ((0xffff & 0xffff & ((0xff & 0x18) ^ (0xffffff00 & ((0xffff & 0xffff & ((0xff & 0xe7) ^ (0xffffff00 & rebx.rebx.0))) ^ (0xffff0000 & var1.0))))) ^ (0xffff0000 & ((0xff & 0xe7) ^ (0xffffff00 & rebx.rebx.0))))))

WWW.ZERONIGHTS.RU

Eq in work

Eq representation of the previous code (after optimization)

rebx.rebx.1 = ((0xff & rebx.rebx.0) ^ (0xffffff00 & ((0xffff & 0xffff & ((0xff & 0x18) ^ (0xffffff00 & ((0xffff & 0xffff & ((0xff & 0xe7) ^ (0xffffff00 & rebx.rebx.0))) ^ (0xffff0000 & var1.0))))) ^ (0xffff0000 & ((0xff & 0xe7) ^ (0xffffff00 & rebx.rebx.0)))))) = rebx.rebx.0

Profit! ☺

WWW.ZERONIGHTS.RU

Eq in work

```
void rustock_sample(UINT32 &rebp, UINT32 &redi, UINT32 &resi)
{
	UINT32 var0, var1, var2;

	var0 = rebp;
	rebp = redi | rebp;
	var1 = redi & var0;
	resi = ~var1;
	var2 = rebp & resi;
	redi = var0 ^ var2;
}
```

A C++ sample of obfuscated code. It was borrowed ☺ from Rustock

WWW.ZERONIGHTS.RU

Eq in work

Eq representation of the previous code (before optimization)

redi.1 = (rebp.0 ^ (((rebp.0 & redi.0) ^ rebp.0 ^ redi.0) & (0xffffffff ^ (rebp.0 & redi.0))))

WWW.ZERONIGHTS.RU

Eq in work

Eq representation of the previous code (after optimization)

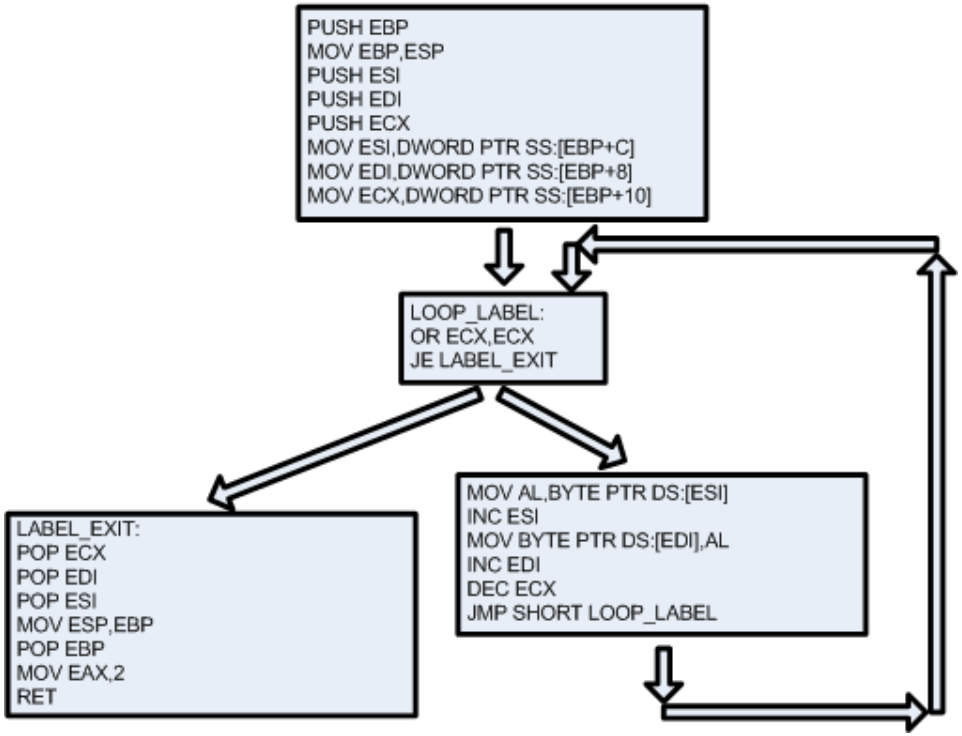
redi.1 = (rebp.0 ^ (((rebp.0 & redi.0) ^ rebp.0 ^ redi.0) & (0xffffffff ^ (rebp.0 & redi.0)))) = redi.0

Profit! ☺

WWW.ZERONIGHTS.RU

Deobfuscation with Eq

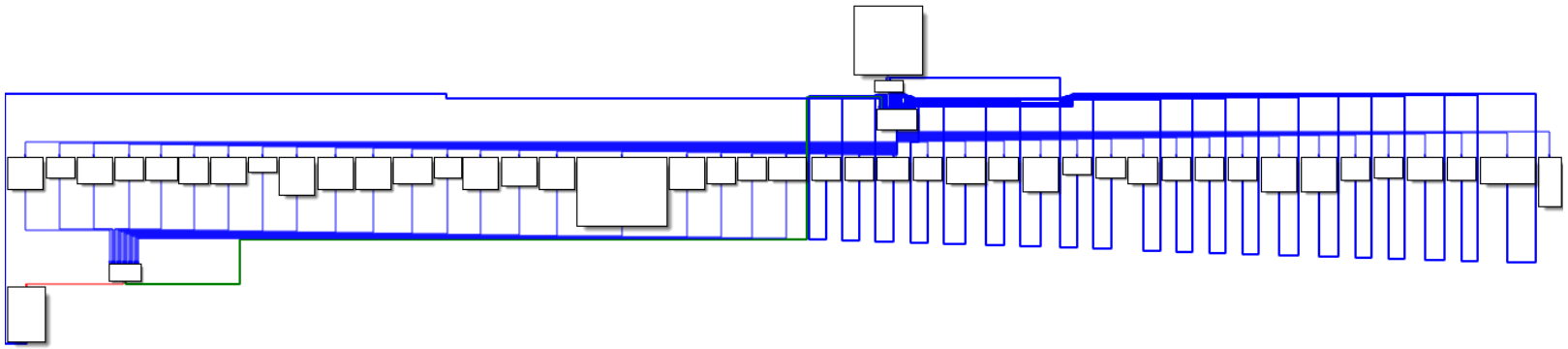
Test function



WWW.ZERONIGHTS.RU

Deobfuscation with Eq

After code virtualization



WWW.ZERONIGHTS.RU

Deobfuscation with Eq

```
UINT32 var0, var1, var2, var3, var4, var5, var6, var7, var8;

        var7 = resp + 0x8;
        var8 = resp;
        resp = resp - 0x10;
        var4 = (UINT32_PTR)*var7;
        var1 = resp + 0x14;
        var5 = (UINT32_PTR)*var1;
        var2 = resp + 0x1c;
        var6 = (UINT32_PTR)*var2;
label1:
        resp = resp - 0x8;
        if( var6 != 0x0 ) goto label2;
        reax = 0x2;
        var0 = (UINT32_PTR)*var8;
        resp = var8 + 0x4;
        goto var0;
label2:
        var3 = var4;
        var4 = var4 + 0x1;
        (UINT8_PTR)*var5 = (UINT8_PTR)*var3;
        var5 = var5 + 0x1;
        var6 = var6 - 0x1;
        resp = resp + 0x8;
        goto label1;
```

The result of deobfuscation

WWW.ZERONIGHTS.RU

Deobfuscation with Eq

- ASProtect
- CodeVirtualizer/Themida/WinLicense
  - old CISC/RISC
  - new Fish/Tiger
- ExeCryptor
- NoobyProtect/SafeEngine
- Tages
- VMProtect
- Some others…

Were deobfuscated successfully ☺

WWW.ZERONIGHTS.RU

Deobfuscation with Eq
Some numbers

Instructions initially	~100
Instructions after obfuscation	~300 000
Instructions after deobfuscation	~200
Code generation time	~4 min
Code deobfuscation time	~2 min
Memory	~300 Mb

WWW.ZERONIGHTS.RU

Obfuscation with Eq

We could use optimization not for deobfuscation only.

What if we could stop optimization process at random step?

WWW.ZERONIGHTS.RU

Obfuscation with Eq

bswap eax

bswap eax

Initial expression
(could be deobfuscated by reverse recursive substitution)

reax.3 = ((((((0xff0000 & reax.0) >> 0x8) & 0xff0000) ^ ((reax.0 >> 0x18) & 0xff0000) ^ ((reax.0 << 0x18) & 0xff0000) ^ (((0xff00 & reax.0) << 0x8) & 0xff0000)) >> 0x8) ^ ((((0xff0000 & reax.0) >> 0x8) ^ (reax.0 >> 0x18) ^ (reax.0 << 0x18) ^ ((0xff00 & reax.0) << 0x8)) >> 0x18) ^ ((((0xff0000 & reax.0) >> 0x8) ^ (reax.0 >> 0x18) ^ (reax.0 << 0x18) ^ ((0xff00 & reax.0) << 0x8)) << 0x18) ^ ((0xff00 & (((0xff0000 & reax.0) >> 0x8) ^ (reax.0 >> 0x18) ^ (reax.0 << 0x18) ^ ((0xff00 & reax.0) << 0x8))) << 0x8))

WWW.ZERONIGHTS.RU

Obfuscation with Eq

Partially optimized expression looks like this:

```
reax.3 = ( ( 0xffff & reax.0 ) ^ ( ( ( 0xff0000 & ( reax.0 << 0x8 ) ) << 0x18 ) ^ ( ( 0xff00 & ( reax.0 >> 0x8 ) ) << 0x18 ) ^ ( ( reax.0 >> 0x18 ) << 0x18 ) ^ ( ( reax.0 << 0x18 ) << 0x18 ) ) ^ ( ( 0xff00 & ( ( 0xff0000 & ( reax.0 << 0x8 ) ) ^ ( 0xff00 & ( reax.0 >> 0x8 ) ) ^ ( reax.0 >> 0x18 ) ^ ( reax.0 << 0x18 ) ) ) << 0x8 ) )
```

WWW.ZERONIGHTS.RU

Obfuscation with Eq

Partially optimized expression looks like this:

```
reax.3 = ( ( 0xffff & reax.0 ) ^ ( ( ( 0xff0000 & ( reax.0 << 0x8 ) ) << 0x18 ) ^ ( ( 0xff00 & ( reax.0 >> 0x8 ) ) << 0x18 ) ^ ( ( reax.0 >> 0x18 ) << 0x18 ) ^ ( ( reax.0 << 0x18 ) << 0x18 ) ) ^ ( ( 0xff00 & ( ( 0xff0000 & ( reax.0 << 0x8 ) ) ^ ( 0xff00 & ( reax.0 >> 0x8 ) ) ^ ( reax.0 >> 0x18 ) ^ ( reax.0 << 0x18 ) ) ) << 0x8 ) )
```

or like this:

```
reax.3 = ( ( 0xffff & reax.0 ) ^ ( 0x0 xor ( ( 0xff00 & ( reax.0 >> 0x8 ) ) << 0x18 ) ^ ( ( reax.0 >> 0x18 ) << 0x18 ) ^ ( ( reax.0 << 0x18 ) << 0x18 ) ) ^ ( ( 0xff00 & ( ( 0xff0000 & ( reax.0 << 0x8 ) ) ^ ( 0xff00 & ( reax.0 >> 0x8 ) ) ^ ( reax.0 >> 0x18 ) ^ ( reax.0 << 0x18 ) ) ) << 0x8 ) )
```

WWW.ZERONIGHTS.RU

Obfuscation with Eq

- Easy to implement
- Hard to deobfuscate using classical compiler theory optimization algorithms
- Hard to deobfuscate using reverse recursive substitution
- No templates and signatures in the obfuscated code

WWW.ZERONIGHTS.RU

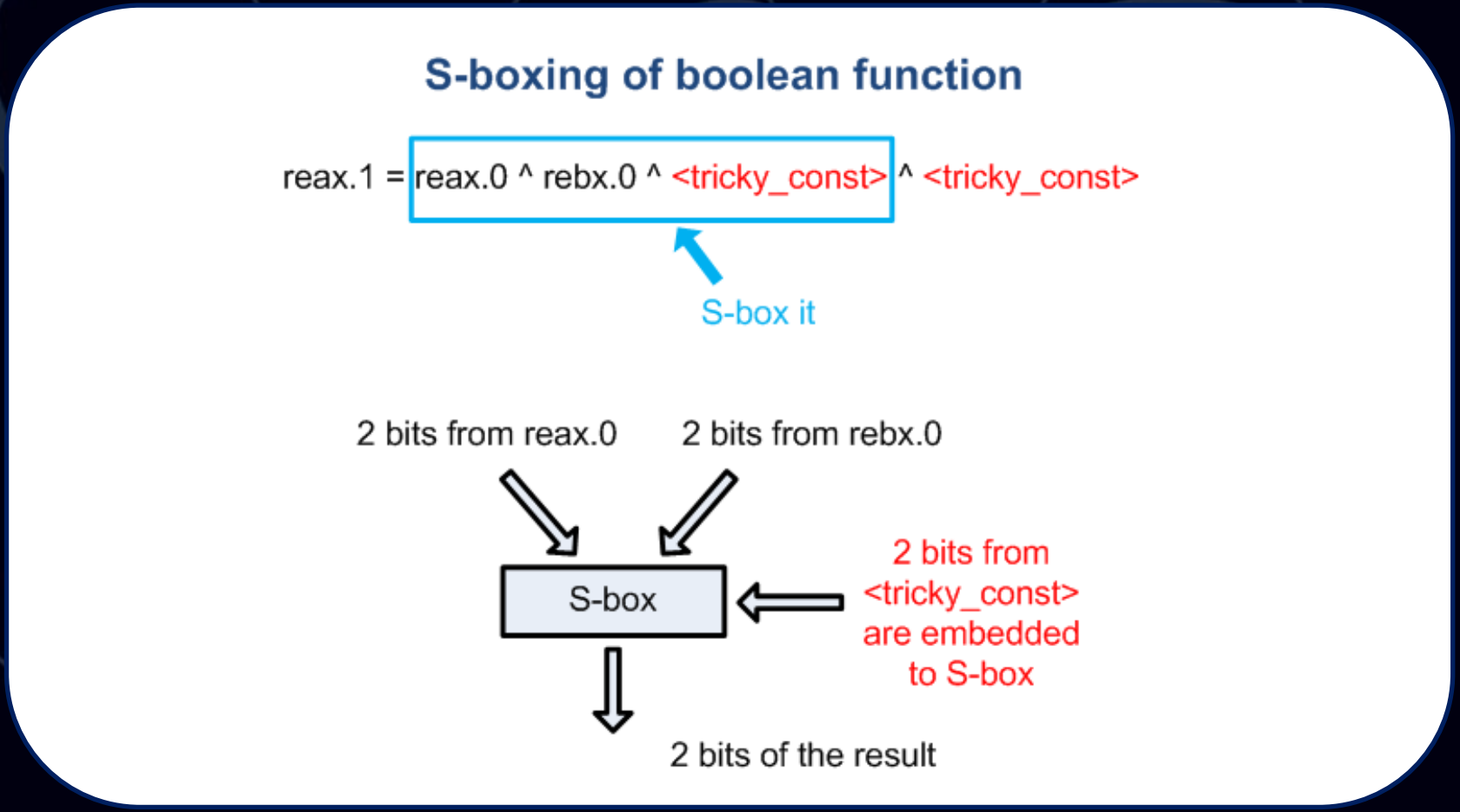
Obfuscation with Eq

But this tricky obfuscation is still weak.
It's possible to deobfuscate these expressions using Eq project or another symbolic equation system.

And we have to go deeper!

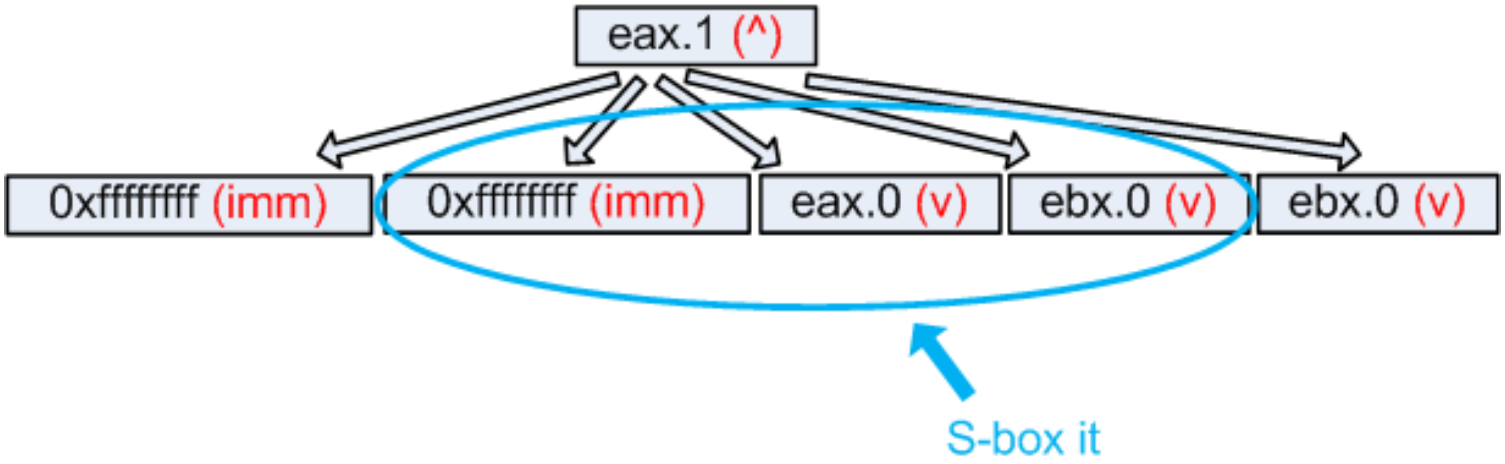
WWW.ZERONIGHTS.RU

Obfuscation with Eq



WWW.ZERONIGHTS.RU

Obfuscation with Eq



Profit! ☺

WWW.ZERONIGHTS.RU

Perspectives

- Obfuscation becomes stronger
  - Complex mathematical expressions are used more frequently
  - Merges with cryptography
- Obfuscation migrates to dark side
  - Protectors are dying
  - Malware market is growing

WWW.ZERONIGHTS.RU

Perspectives

- Obfuscation becomes undetectable
  - Mimicry methods are improved
  - Obfuscators try to avoid method of recursive substitutions
  - Obfuscators use well-known high-level platforms

- LLVM becomes a generic platform for creating obfuscators

WWW.ZERONIGHTS.RU

Questions

ZERO NIGHTS

WWW.ZERONIGHTS.RU