For Realz Mode!

Reversing an MBR from the CSAW CTF

(and how to write a keygen with a SAT solver)

t0x0 @numinit

V A P O R S E C

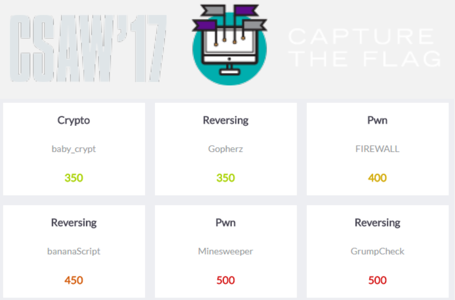
2017-10-11

About us

- @numinit
  - Software developer by day
  - Tinkerer on everything by night
  - First Def Con and Toorcon experiences this year
  - Doing CSAW CTF for a while
- t0x0 (@t0x0pg)
  - Writes lots of interpreted code
  - Lives mostly in Windows world
  - Jack of all trades, master of none (so far)
  - Obsessively curious

What's CSAW?

An annual security capture the flag run by New York Polytechnic that contains challenges with a wide range of difficulties, attracting everyone from undergraduates to well-known teams like PPP



Who are V A P O R S E C ?

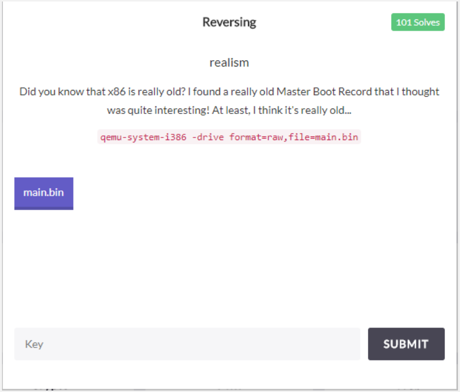
- A new A E S T H E T I C San Diego CTF team
- Had 10 participants during the CSAW CTF
- Got 15th in CSAW industry professional bracket, which is cool I guess

Outline

1. Introduction
2. Reversing the MBR
3. Finding the flag

RE400 what?...

- Challenge worth 400 points
- Reverse Engineering category
- We get some hints right away...
  - This is an MBR
  - ...from an x86 system



A place to start...

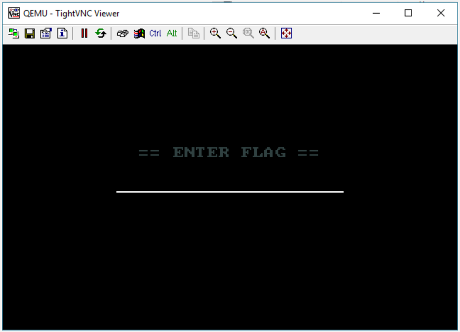
- Wikipedia, of course!

A place to start...

- Wikipedia, of course!
  - 512 bytes
  - MBR signature: 55 AA
  - "expected to contain real mode machine language instructions"
  - little-endian
  - loads at 0000:7C00

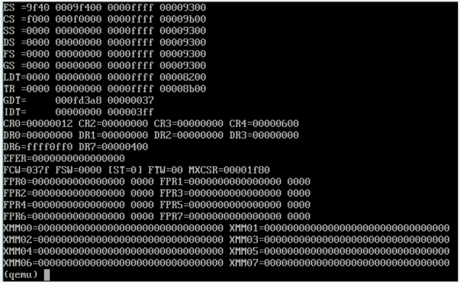
Tool Time!...

- qemu (gift wrapped)
  - -s (gdb)
  - -S (suspend)
  - -vnc:1



Tool Time!...

- qemu (gift wrapped)
    - QEMU/Monitor
        - info registers
        - system reset



A quick recap...

- We have an x86 boot sector that's asking us for the flag

A quick recap...

- We have an x86 boot sector that's asking us for the flag
- The flag is clearly in the boot sector SOMEWHERE...

A quick recap...

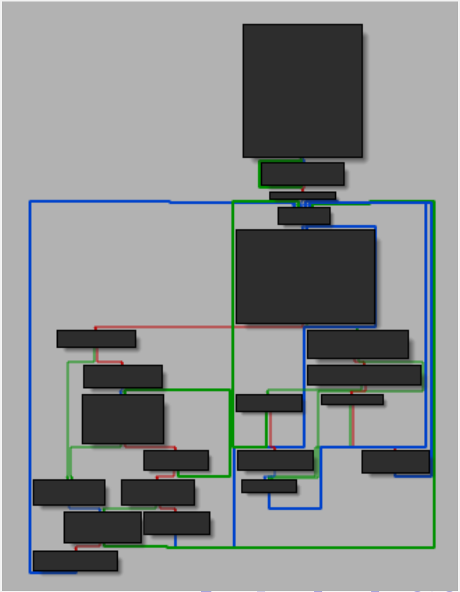
- We have an x86 boot sector that's asking us for the flag
- The flag is clearly in the boot sector SOMEWHERE...
  - ...but not in plaintext, because this is a 400 point challenge and that would be too easy

A quick recap...

- We have an x86 boot sector that's asking us for the flag
- The flag is clearly in the boot sector SOMEWHERE...
  - ...but not in plaintext, because this is a 400 point challenge and that would be too easy
- So, where is it?

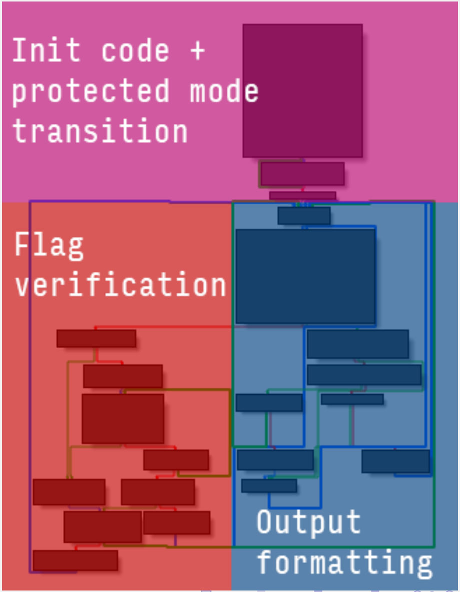
Reversing the boot sector

- Open the file in your favorite disassembler (e.g. IDA); rebase at 0x7c00



Reversing the boot sector

- Open the file in your favorite disassembler (e.g. IDA); rebase at `0x7c00`
- We can visally pick out three sections:
  - Init code (responsible for the protected mode transition)
  - Display code (identifiable by a large number of `int` instructions)
  - Some other code that uses Intel SSE2 instructions



First leads

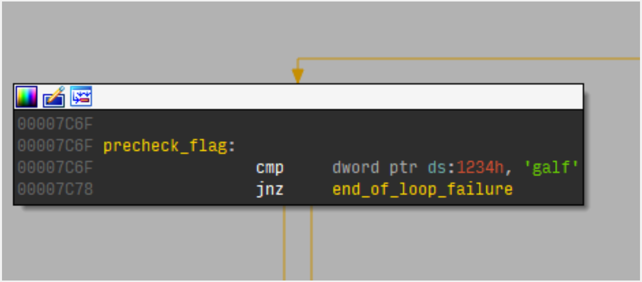
- The program asks you to enter 20 characters, and immediately breaks out if the first 4 characters aren't 'flag' after you enter character #20

First leads

- The program asks you to enter 20 characters, and immediately breaks out if the first 4 characters aren't 'flag' after you enter character #20
- We can verify this in a debugger after eyeballing the code

First leads

- The program asks you to enter 20 characters, and immediately breaks out if the first 4 characters aren't 'flag' after you enter character #20
- We can verify this in a debugger after eyeballing the code



As it turns out...

After several hours of staring at x86 assembly

- The flag is hashed using a custom algorithm implemented with Intel SSE instructions

As it turns out...

After several hours of staring at x86 assembly

- The flag is hashed using a custom algorithm implemented with Intel SSE instructions
- *Ugh...*

As it turns out...

After several hours of staring at x86 assembly

- The flag is hashed using a custom algorithm implemented with Intel SSE instructions
- *Ugh...*
- We have to find an input to the hash algorithm that hashes to the same value that's stored in the boot sector

Suddenly, SSE2 instructions

"My Intel CPU can do *that?*"

The author of the challenge decided to use a bunch of obscure x86 SSE2 instructions to force us to trawl through Intel documentation

Note

SSE2 instructions operate on XMM registers, which are 128 bits (16 bytes) wide.

Suddenly, SSE2 instructions

"My Intel CPU can do *that?*"

The author of the challenge decided to use a bunch of obscure x86 SSE2 instructions to force us to trawl through Intel documentation

- movaps
- andps
- pshufd
- psadbw

Note

SSE2 instructions operate on XMM registers, which are 128 bits (16 bytes) wide.

Suddenly, SSE2 instructions

"My Intel CPU can do *that?*"

The author of the challenge decided to use a bunch of obscure x86 SSE2 instructions to force us to trawl through Intel documentation

- movaps: Moves to/from/between XMM registers
- andps: Performs bitwise AND between XMM registers
- pshufd: Reorders 32-bit words in XMM registers
- psadbw: Sums absolute values of differences between bytes (it's as crazy as it sounds)

Note

SSE2 instructions operate on XMM registers, which are 128 bits (16 bytes) wide.

Suddenly, SSE2 instructions

"My Intel CPU can do *that?*"

The author of the challenge decided to use a bunch of obscure x86 SSE2 instructions to force us to trawl through Intel documentation

- movaps: Moves to/from/between XMM registers
- andps: Performs bitwise AND between XMM registers
- pshufd: Reorders 32-bit words in XMM registers
- psadbw: Sums absolute values of differences between bytes (it's as crazy as it sounds)

Note

SSE2 instructions operate on XMM registers, which are 128 bits (16 bytes) wide.

Question

Why are these instructions useful for writing a hash function (even if it's a bad hash function that we can break?)

Packed Sum of Absolute Differences of Bytes in Word

That's a mouthful...

xmm0:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Packed Sum of Absolute Differences of Bytes in Word

That's a mouthful...

xmm0:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

(minus)

xmm1:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

It's hard to go back...

If you just have the result, you have to find two sets of eight bytes where the absolute values of their differences sum to 64 (and there are many)

Packed Sum of Absolute Differences of Bytes in Word

That's a mouthful...

xmm0:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

(minus)

xmm1:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

$\downarrow$

xmm0:

-15	-13	-11	-9	-7	-5	-3	-1	1	3	5	7	9	11	13	15

It's hard to go back...

If you just have the result, you have to find two sets of eight bytes where the absolute values of their differences sum to 64 (and there are many)

Packed Sum of Absolute Differences of Bytes in Word

That's a mouthful...

xmm0:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

(minus)

xmm1:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

$\downarrow$

xmm0:	−15	−13	−11	−9	−7	−5	−3	−1	1	3	5	7	9	11	13	15

(absolute values)

xmm0:	15	13	11	9	7	5	3	1	1	3	5	7	9	11	13	15

It's hard to go back...

If you just have the result, you have to find two sets of eight bytes where the absolute values of their differences sum to 64 (and there are many)

Packed Sum of Absolute Differences of Bytes in Word

That's a mouthful...

xmm0:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

(minus)

xmm1:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

↓

xmm0:

−15	−13	−11	−9	−7	−5	−3	−1	1	3	5	7	9	11	13	15

(absolute values)

xmm0:

15	13	11	9	7	5	3	1	1	3	5	7	9	11	13	15

(16-bit sums)

xmm0:

0	0	0	0	0	0	64	0	0	0	0	0	0	0	64	0

It's hard to go back...

If you just have the result, you have to find two sets of eight bytes where the absolute values of their differences sum to 64 (and there are many)

SAT solvers to the rescue!

How to master Sudoku without memorizing strategies

"I think a serious case can be made that the decline in the American economy can be blamed on the sapping of the mental energy and productivity of the American workforce that sudoku addiction alone has wrought"

– Some Slate writer

SAT solvers to the rescue!

How to master Sudoku without memorizing strategies

"I think a serious case can be made that the decline in the American economy can be blamed on the sapping of the mental energy and productivity of the American workforce that sudoku addiction alone has wrought"

– Some Slate writer

Notice anything strange about this puzzle?

d	c	6	1	9	0	e	b	5	7	3	2	4	8	f	a
5	8	a	4	f	c	2	d	9	0	b	e	3	7	6	1
2	b	9	7	5	3	6	1	4	8	a	f	d	0	e	c
0	3	f	e	4	a	8	7	d	c	6	1	9	b	2	5
8	d	c	6	1	9	3	2	b	5	e	7	0	4	a	f
b	1	5	0	d	8	c	6	f	a	9	4	7	2	3	e
9	a	4	f	7	5	0	e	2	3	1	8	b	d	c	6
7	2	e	3	b	4	a	f	0	d	c	6	1	9	5	8
e	4	d	c	6	1	9	0	a	b	2	5	8	f	7	3
1	9	8	2	3	7	5	4	e	6	f	0	a	c	d	b
f	6	7	a	2	d	b	c	1	9	8	3	5	e	0	4
3	5	0	b	a	e	f	8	7	4	d	c	6	1	9	2
a	e	3	d	c	6	1	9	8	f	7	b	2	5	4	0
6	0	2	5	e	b	d	3	c	1	4	9	f	a	8	7
c	7	1	9	8	f	4	5	6	2	0	a	e	3	b	d
4	f	b	8	0	2	7	a	3	e	5	d	c	6	1	9

SAT solvers to the rescue!

How to master Sudoku without memorizing strategies

"I think a serious case can be made that the decline in the American economy can be blamed on the sapping of the mental energy and productivity of the American workforce that sudoku addiction alone has wrought"

– Some Slate writer

d	c	6	1	9	0	e	b	5	7	3	2	4	8	f	a
5	8	a	4	f	c	2	d	9	0	b	e	3	7	6	1
2	b	9	7	5	3	6	1	4	8	a	f	d	0	e	c
0	3	f	e	4	a	8	7	d	c	6	1	9	b	2	5
8	d	c	6	1	9	3	2	b	5	e	7	0	4	a	f
b	1	5	0	d	8	c	6	f	a	9	4	7	2	3	e
9	a	4	f	7	5	0	e	2	3	1	8	b	d	c	6
7	2	e	3	b	4	a	f	0	d	c	6	1	9	5	8
e	4	d	c	6	1	9	0	a	b	2	5	8	f	7	3
1	9	8	2	3	7	5	4	e	6	f	0	a	c	d	b
f	6	7	a	2	d	b	c	1	9	8	3	5	e	0	4
3	5	0	b	a	e	f	8	7	4	d	c	6	1	9	2
a	e	3	d	c	6	1	9	8	f	7	b	2	5	4	0
6	0	2	5	e	b	d	3	c	1	4	9	f	a	8	7
c	7	1	9	8	f	4	5	6	2	0	a	e	3	b	d
4	f	b	8	0	2	7	a	3	e	5	d	c	6	1	9

Notice anything strange about this puzzle?

Stranger still: a SAT solver created it from thin air

What's a SAT (or SMT) solver?

- SAT solvers

What's a SAT (or SMT) solver?

- SAT solvers
    - Only operate on Boolean logic

What's a SAT (or SMT) solver?

- SAT solvers
  - Only operate on Boolean logic
  - Take large sets of Boolean CNFs (e.g. $(v_1 \vee v_2 \vee v_3) \wedge (v_4 \vee v_5 \vee v_6)$)

What's a SAT (or SMT) solver?

- SAT solvers
  - Only operate on Boolean logic
  - Take large sets of Boolean CNFs (e.g. $(v_1 \vee v_2 \vee v_3) \wedge (v_4 \vee v_5 \vee v_6)$)
  - Try to come up with inputs so the formula returns true (or fail and report that the formula is "unsatisfiable", which means that it's impossible for the formula to be true)

What's a SAT (or SMT) solver?

- SAT solvers
  - Only operate on Boolean logic
  - Take large sets of Boolean CNFs (e.g. $(v_1 \vee v_2 \vee v_3) \wedge (v_4 \vee v_5 \vee v_6)$)
  - Try to come up with inputs so the formula returns true (or fail and report that the formula is "unsatisfiable", which means that it's impossible for the formula to be true)
- SMT solvers

What's a SAT (or SMT) solver?

- SAT solvers
  - Only operate on Boolean logic
  - Take large sets of Boolean CNFs (e.g. $(v_1 \vee v_2 \vee v_3) \wedge (v_4 \vee v_5 \vee v_6)$)
  - Try to come up with inputs so the formula returns true (or fail and report that the formula is "unsatisfiable", which means that it's impossible for the formula to be true)
- SMT solvers
  - "Satisfiability modulo theories" - a fancy term for a piece of software that uses a SAT solver to act as a theorem prover

What's a SAT (or SMT) solver?

- SAT solvers
  - Only operate on Boolean logic
  - Take large sets of Boolean CNFs (e.g. $(v_1 \vee v_2 \vee v_3) \wedge (v_4 \vee v_5 \vee v_6)$)
  - Try to come up with inputs so the formula returns true (or fail and report that the formula is "unsatisfiable", which means that it's impossible for the formula to be true)
- SMT solvers
  - "Satisfiability modulo theories" - a fancy term for a piece of software that uses a SAT solver to act as a theorem prover
  - Takes standard mathematical operations, bit operations, etc, and converts them to Boolean CNFs that a SAT solver can work with

What's a SAT (or SMT) solver?

- SAT solvers
  - Only operate on Boolean logic
  - Take large sets of Boolean CNFs (e.g. $(v_1 \vee v_2 \vee v_3) \wedge (v_4 \vee v_5 \vee v_6)$)
  - Try to come up with inputs so the formula returns true (or fail and report that the formula is "unsatisfiable", which means that it's impossible for the formula to be true)
- SMT solvers
  - "Satisfiability modulo theories" - a fancy term for a piece of software that uses a SAT solver to act as a theorem prover
  - Takes standard mathematical operations, bit operations, etc, and converts them to Boolean CNFs that a SAT solver can work with
  - Again, tries to come up with inputs so that the formula is satisfiable

What's a SAT (or SMT) solver?

- SAT solvers
  - Only operate on Boolean logic
  - Take large sets of Boolean CNFs (e.g. $(v_1 \lor v_2 \lor v_3) \land (v_4 \lor v_5 \lor v_6)$)
  - Try to come up with inputs so the formula returns true (or fail and report that the formula is "unsatisfiable", which means that it's impossible for the formula to be true)
- SMT solvers
  - "Satisfiability modulo theories" - a fancy term for a piece of software that uses a SAT solver to act as a theorem prover
  - Takes standard mathematical operations, bit operations, etc, and converts them to Boolean CNFs that a SAT solver can work with
  - Again, tries to come up with inputs so that the formula is satisfiable
- Symbolic execution engines

What's a SAT (or SMT) solver?

- SAT solvers
  - Only operate on Boolean logic
  - Take large sets of Boolean CNFs (e.g. $(v_1 \vee v_2 \vee v_3) \wedge (v_4 \vee v_5 \vee v_6)$)
  - Try to come up with inputs so the formula returns true (or fail and report that the formula is "unsatisfiable", which means that it's impossible for the formula to be true)
- SMT solvers
  - "Satisfiability modulo theories" - a fancy term for a piece of software that uses a SAT solver to act as a theorem prover
  - Takes standard mathematical operations, bit operations, etc, and converts them to Boolean CNFs that a SAT solver can work with
  - Again, tries to come up with inputs so that the formula is satisfiable
- Symbolic execution engines
  - Convert CPU instructions into a SMT solver language

In case this all seemed too abstract...

SMT solvers let you write a type of SQL that can be used to solve certain hard* problems

In case this all seemed too abstract...

SMT solvers let you write a type of SQL that can be used to solve certain hard* problems

* hard, meaning “NP-complete”

In case this all seemed too abstract...

SMT solvers let you write a type of SQL that can be used to solve certain hard* problems

* hard, meaning "NP-complete"

The SMT query itself can be constructed from a language like Python, C, or even raw x86 assembly

Modern SMT solvers and symbolic execution engines

- Z3 (SMT)
  - Open source, developed by Microsoft
  - Has its own SQL-like language, with Python bindings
  - Contains its own SAT solver

Modern SMT solvers and symbolic execution engines

- Z3 (SMT)
  - Open source, developed by Microsoft
  - Has its own SQL-like language, with Python bindings
  - Contains its own SAT solver
- KLEE (symbolic execution)
  - Used to instrument code that can be recompiled to LLVM IR
  - C bindings
  - Z3, Kleaver, or SMTLib used as a backend

Modern SMT solvers and symbolic execution engines

- Z3 (SMT)
  - Open source, developed by Microsoft
  - Has its own SQL-like language, with Python bindings
  - Contains its own SAT solver
- KLEE (symbolic execution)
  - Used to instrument code that can be recompiled to LLVM IR
  - C bindings
  - Z3, Kleaver, or SMTLib used as a backend
- Ponce (symbolic execution)
  - An IDA Pro plugin for x86 and x86_64 instruction sets
  - Z3 used as a backend

Modern SMT solvers and symbolic execution engines

- Z3 (SMT)
  - Open source, developed by Microsoft
  - Has its own SQL-like language, with Python bindings
  - Contains its own SAT solver
- KLEE (symbolic execution)
  - Used to instrument code that can be recompiled to LLVM IR
  - C bindings
  - Z3, Kleaver, or SMTLib used as a backend
- Ponce (symbolic execution)
  - An IDA Pro plugin for x86 and x86_64 instruction sets
  - Z3 used as a backend

We'll be mainly focusing on using Z3, since all the other tools use it under the hood (and it's awesome)

Back to our sudoku...

How do you create a hexadoku puzzle out of thin air with Z3?

Back to our sudoku...

How do you create a hexadoku puzzle out of thin air with Z3?

Creating the board

```
import z3

# Define a 16x16 Sudoku
order = 4
side = order ** 2

# Create a solver
smt = z3.Solver()

cells = [list() for row in range(side)]
for r in range(side):
    for c in range(side):
        cells[r].append(z3.Int('cell_%d_%d' % (r, c)))
```

Telling z3 how to play sudoku

We'll define our first rule: each cell has to be between 0 and 15

Telling z3 how to play sudoku

We'll define our first rule: each cell has to be between 0 and 15

The most basic of Sudoku rules

```
for r in range(side):
    for c in range(side):
        # Each cell must be between 0 and 15
        smt.add(z3.And(cells[r][c] >= 0, cells[r][c] < side))
```

Telling z3 how to play sudoku

Now, the rules everyone knows

Telling z3 how to play sudoku

Now, the rules everyone knows

Rows and columns

```
# Rows must have unique values
for r in range(side):
    this_row = [cells[r][c] for c in range(side)]
    smt.add(z3.Distinct(*this_row))

# Columns must have unique values
for c in range(side):
    this_col = [cells[r][c] for r in range(side)]
    smt.add(z3.Distinct(*this_col))
```

Mini-boxes are slightly more complicated

Telling z3 how to play sudoku

Mini-boxes are slightly more complicated

Mini-boxes

```
# Each mini-box must have unique values
for r in range(0, side, order):
    for c in range(0, side, order):
        selected = []
        for i in range(box_size):
            for j in range(box_size):
                selected.append(cells[r + i][c + j])

        smt.add(z3.Distinct(*selected))
```

Telling z3 how to play *our version of* sudoku

Now for the initial values
... dc858 isn't as sudoku-able :-(

Telling z3 how to play *our version of* sudoku

Now for the initial values
... dc858 isn't as sudoku-able :-(

Initial values

```
# Now, define the initial values
smt.add(cells[0][0] == 0xd)
smt.add(cells[0][1] == 0xc)
smt.add(cells[0][2] == 0x6)
smt.add(cells[0][3] == 0x1)
smt.add(cells[0][4] == 0x9)


# ... etc.
```

Telling z3 to start solving, and printing the result

This will take a little bit of time

Telling z3 to start solving, and printing the result

This will take a little bit of time

Start solving!

```
if smt.check() == z3.unsat:
    print('Sudoku is impossible to solve :(')
else:
    model = smt.model()

    # Retrieve each cell from the model as a python long
    result = [list() for row in range(side)]
    for r in range(side):
        for c in range(side):
            cell = model[r][c]
            print('%x ' % model[cell].as_long(), end='')
        print()
```

The results...

d	c	6	1	9	0	e	b	5	7	3	2	4	8	f	a
5	8	a	4	f	c	2	d	9	0	b	e	3	7	6	1
2	b	9	7	5	3	6	1	4	8	a	f	d	0	e	c
0	3	f	e	4	a	8	7	d	c	6	1	9	b	2	5
8	d	c	6	1	9	3	2	b	5	e	7	0	4	a	f
b	1	5	0	d	8	c	6	f	a	9	4	7	2	3	e
9	a	4	f	7	5	0	e	2	3	1	8	b	d	c	6
7	2	e	3	b	4	a	f	0	d	c	6	1	9	5	8
e	4	d	c	6	1	9	0	a	b	2	5	8	f	7	3
1	9	8	2	3	7	5	4	e	6	f	0	a	c	d	b
f	6	7	a	2	d	b	c	1	9	8	3	5	e	0	4
3	5	0	b	a	e	f	8	7	4	d	c	6	1	9	2
a	e	3	d	c	6	1	9	8	f	7	b	2	5	4	0
6	0	2	5	e	b	d	3	c	1	4	9	f	a	8	7
c	7	1	9	8	f	4	5	6	2	0	a	e	3	b	d
4	f	b	8	0	2	7	a	3	e	5	d	c	6	1	9

The power of SMT solvers

Maybe familiar to anyone who's tried Prolog

We only had to define the constraints of the puzzle we wanted, and Z3 found one satisfying those constraints

Constraint solving our boot sector

"Hey, Z3, I want you to find me input values that, after being put through this crazy operation, end up being equal to the ones stored in the boot sector"

xmm0:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

(minus)

xmm1:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

↓

xmm0:	−15	−13	−11	−9	−7	−5	−3	−1	1	3	5	7	9	11	13	15

(absolute values)

xmm0:	15	13	11	9	7	5	3	1	1	3	5	7	9	11	13	15

(16-bit sums)

xmm0:	0	0	0	0	0	0	64	0	0	0	0	0	0	0	64	0