SMT-решатели в ИБ

введение в использование SMT
на примере анализа защиты
программного обеспечения

Defcon Russia (DCG #7812)

whoami

- системный программист
- реверс-инженер
- исследователь безопасности в Digital Security

Agenda

- об SMT простыми словами
  - Z3 - SMT решатель от Microsoft Research
- SMT и ИБ
  - поиск уязвимостей
  - генерация эксплойтов
  - анализ механизмов защиты
- автоматическое создание кейгенов

Что такое SMT-решатель?

- SMT-решатель это программа
- вы можете задать ей какой-нибудь вопрос:
  - «в чём смысл жизни?»
- она попробует дать ответ:
  - «**42**»

- http://en.wikipedia.org/wiki/Satisfiability_Modulo_Theories
- http://fm.csl.sri.com/SSFT12/introduction.pdf

Как задавать вопрос?

- вопрос это логическая формула
  - $b + 2 = c$, $f(read(write(a,b,3), c-2)) \neq f(c-b+1)$
- SMT-LIB: язык для выражения формулы
  - http://www.smtlib.org/
- все решатели должны понимать этот язык

- http://www.grammatech.com/resources/smt/SMTLIBTutorial.pdf

Какой SMT-решатель выбрать ?

- SMT-решателей много (> 20):
  - Ario, CVC3, MathSAT, STP, Yices, z3(Microsoft), etc.
- мой выбор - z3 от Microsoft:
  - высокая производительность (победитель SMT-COMP 2011)
  - API для Python, OCaml, C, C++, .NET, Java
  - доступен on-line, имеет клЁвые туториалы
  - используется во многих разработках Microsoft
- http://rise4fun.com/z3/tutorial
- http://rise4fun.com/z3py/tutorial

ВЫП или не ВЫП ?
Задаём вопрос.

- структура формулы
  - объявление
  - предусловие
  - постусловие
- ответ: sat или unsat
  - выполнимость
  - вылидность
- !!! модель данных !!!

z3 Microsoft Research

Is this formula satisfiable?

```
(declare-const work Int)
(declare-const sleep Int)
(declare-const fun Int)

(assert (>= work 40))  ; работать как минимум 40 часов в неделю
(assert (>= sleep 42)) ; спать хотя бы 6 часов в день
(assert (>= fun work)) ; баланс между работой и весельем

(assert (= (+ work (+ sleep fun)) 168)) ; 168 часов в неделе

(check-sat) ; просим z3 решить формулу
(get-model) ; просим z3 вывести модель
```

tutorial home video permalink

'►' shortcut: Alt+B

```
sat
(model
  (define-fun sleep () Int
    42)
  (define-fun fun () Int
    63)
  (define-fun work () Int
    63)
)
```

- Taint Nobody Got Time for Crash Analysis

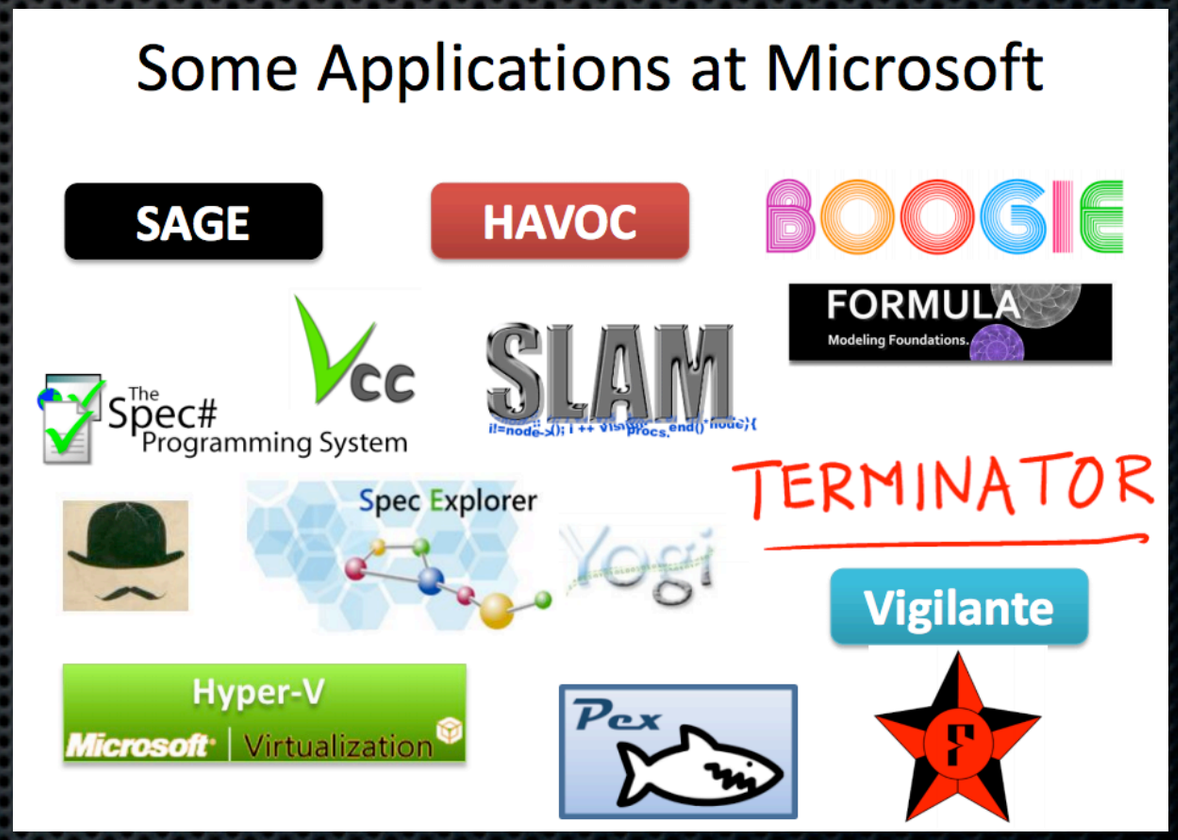
http://rise4fun.com/Z3/pLpMc

Что нужно знать об SMT

- Математическая точность
  - оставьте все сомнения
- Выразительная сила
  - подходит для моделирования алгоритмов и CPU
- Предоставляют модель данных
  - замечательное свойство
- Существуют эффективные реализации

Области применения SMT

- планирование
- системная биология
- искусственный интеллект
- разработка ПО
- etc.



Применение SMT в ИБ

- **Основная идея:**
  - ***Преобразовать участок кода в SMT-формулу, затем применить точный математический метод чтобы проверить определенное свойство этого кода.***

Поиск уязвимостей и SMT

- Статический анализ на уязвимости
  - автоматический (VCC, HAVOC, Boogie)
  - ручной (review)
- Fuzzing
  - увеличение покрытия кода

Review и SMT

- функция аутентификации
- nresp - контролируется пользователем
- nresp - используется для выделения памяти
- packet_get_string () - принимает данные из сети

```
/*
    Целочисленное переполнение в OpenSSH 3.1
*/

void input_userauth_info_response()
{
    //...
    u_int nresp;

    //...
    nresp = packet_get_int();
    if (nresp > 0) {
        response = xmalloc(nresp * sizeof(char*));

        for (i = 0; i < nresp; i++)
            response[i] = packet_get_string(NULL);
    }
    packet_check_eom();
}
```

Review и SMT

```c
/*
    Целочисленное переполнение в OpenSSH 3.1
*/

void input_userauth_info_response()
{
    //...
    u_int nresp;

    //...
    nresp = packet_get_int();
    if (nresp > 0) {
        response = xmalloc(nresp * sizeof(char*));

        for (i = 0; i < nresp; i++)
            response[i] = packet_get_string(NULL);
    }
    packet_check_eom();
}
```

z3 Microsoft Research

Is this formula satisfiable?

```
(declare-const sizeof (_ BitVec 32))
(declare-const nresp  (_ BitVec 32))
(declare-const mult   (_ BitVec 32))

(assert (= sizeof (_ bv4 32)))          ; sizeof (char*) = 4
(assert (= mult (bvmul nresp sizeof))) ; nresp*sizeof

(assert (bvugt nresp (_ bv0 32)))       ; nresp > 0
(assert (bvult mult nresp))             ; nresp*sizeof < nresp
(assert (= mult (_ bv256 32)))          ; nresp*sizeof = 256

(check-sat)
(get-model)
```

tutorial

home video permalink

'►' shortcut: Alt+B

```
sat
(model
  (define-fun nresp () (_ BitVec 32)
    #x40000040)
```

http://rise4fun.com/Z3/1Tp0g

Fuzzing и SMT

- Чем больше покрытие кода, тем лучше!
- SMT-решатели - часть продвинутых систем фаззинга, наряду с:
  - Symbolic/Concolic execution
  - Taint analysis
  - Dynamic Binary Instrumentation
- используются для удовлетворения ограничений
- All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution
- Dynamic Binary Instrumentation в ИБ

Fuzzing и SMT

```
void foo (int a, int b, int c)
{
    if (a > 0 && b < 20){
        if (c > a && c < b){
            vuln_func();
            }
    }
}
```

z3 Microsoft Research

Is this formula satisfiable?

```
(declare-const a Int)
(declare-const b Int)
(declare-const c Int)

(assert (and (> a 0) (< b 20)))  ; (a > 0) && (b < 20)
(assert (and (> c a) (< c b)))   ; (c > a) && (c < b)

(check-sat)
(get-model)
```

tutorial

home video permalink

'►' shortcut: Alt+B

```
sat
(model
  (define-fun b () Int
    3)
  (define-fun a () Int
    1)
  (define-fun c () Int
    2)
)
```

http://rise4fun.com/Z3/qnX6

Ссылки по теме

- VCC, HAVOC, Boogie
- Sean Heelan's Blog
- Immunity Debugger\Libs\x86smt
- Concolic execution - Taint analysis with Valgrind and constraints path solver with Z3
- All You Ever Wanted to Know About Dynamic Taint Analysis and Forward Symbolic Execution
- Fuzzgrind

Генерация эксплойтов и SMT-решатели

- Автоматическая генерация ввода для перехвата потока выполнения
- Генерация полезной нагрузки
  - ROP-only шелл-код

Генерация ввода

- снимаем трассу до уязвимого кода
- преобразуем трассу во множество ограничений
  - F_{reach}
- формируем условия эксплуатации
  - $F_{exploit} = C_{val} \cup C_{addr}$
- решаем $solve(F_{reach} \cup F_{exploit})$
- SMT-решатель определит необходимый ввод

- http://dcc.fceia.unr.edu.ar/~ggrieco/extended_abstract_CSTVA13.pdf

Генерация ввода

```
00:     call
01:     add eax, ebx    ; t0 = eax + ebx
02:     jnz somewhere   ; if zf == 0 then somewhere
03:     add ebp,ecx     ; t1 = ebp + ecx
04:     mov [ebp], eax  ; Mem[t1] = eax
    ;...
somewhere:
    ;...
99:     ret
```

- F_{reach} = {t0 = eax + ebx, zf == 1}
- $C_{\underline{val}}$ = { eax = 0xdeadbeef }
- C_{addr} = {t1 = ebp + 4, t1 = ebp + ecx}
- $F_{exploit} = C_{val} \cup C_{addr}$

Генерация ввода

- F_{reach} =
  {t0 = eax + ebx, zf == 1}
- C_{val} =
  { eax = 0xdeadbeef }
- C_{addr} =
  {t1 = ebp + 4, t1 = ebp + ecx}
- $F_{exploit} = C_{val} \cup C_{addr}$
- **Input ={eax = 0xdeadbeef, ebx = 0x21524111, ecx = 4}**

z3 Microsoft Research

Is this formula satisfiable?

```
(declare-const t0 (_ BitVec 32))
(declare-const t1 (_ BitVec 32))

(declare-const eax (_ BitVec 32))
(declare-const ebx (_ BitVec 32))
(declare-const ecx (_ BitVec 32))
(declare-const ebp (_ BitVec 32))

; Freach = {zf = 1, t0 = eax + ebx}
(assert (= t0 (bvadd eax ebx)))     ; t0 = eax + ebx
(assert (= t0 #x00000000))          ; zf = 1

;Cval = { eax = 0xDEADBEEF }
(assert (= eax #xdeadbeef))         ; eax = 0xDEADBEEF

;Caddr = { t1 = ebp + 4, t1 = ebp + ecx}
(assert
  (and
  (= t1 (bvadd ebp #x00000004))   ; t1 = ebp + 4
  (= t1 (bvadd ebp ecx)))         ; t2 = ebp + ecx
)
```

tutorial home video permalink

'►' shortcut: Alt+B

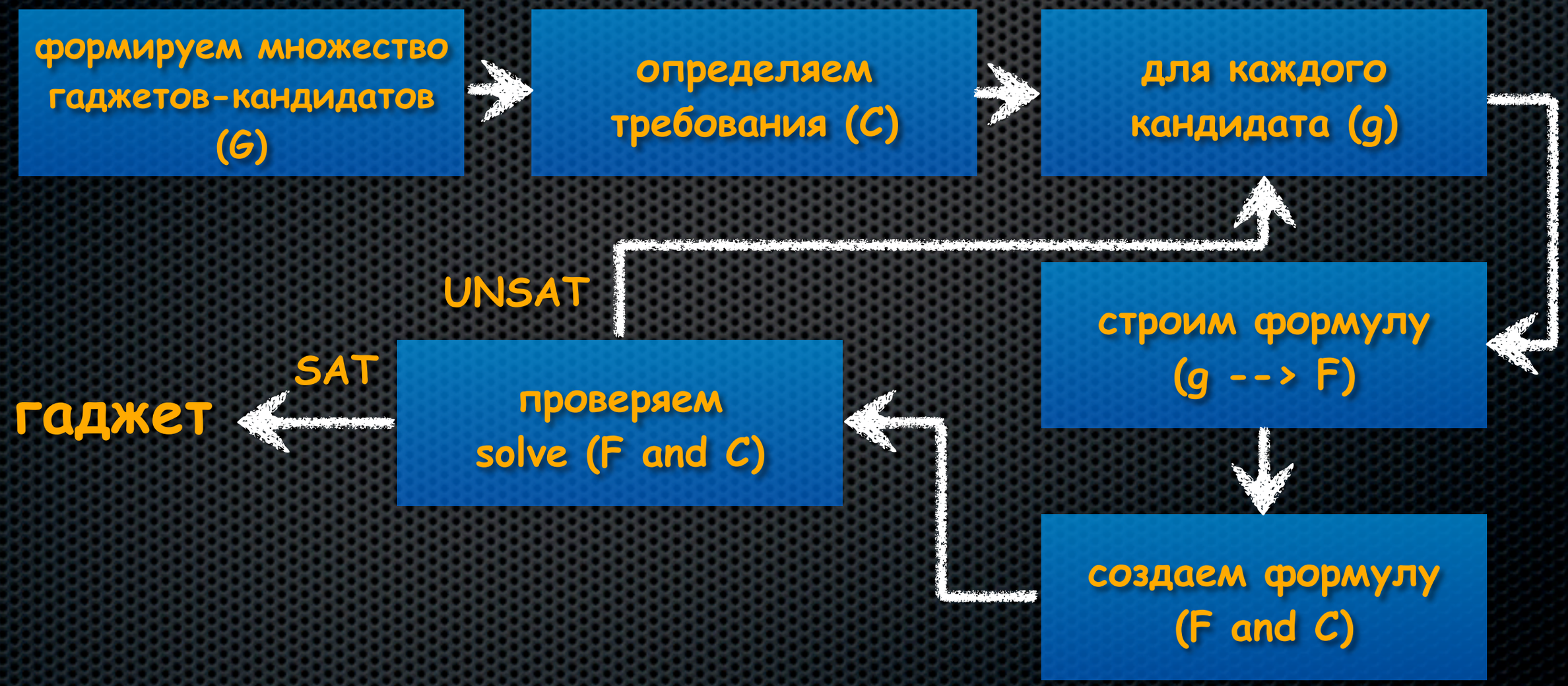
```
sat
(model
  (define-fun ecx () (_ BitVec 32)
    #x00000004)
  (define-fun ebp () (_ BitVec 32)
    #x00000000)
  (define-fun t1 () (_ BitVec 32)
    #x00000004)
  (define-fun eax () (_ BitVec 32)
    #xdeadbeef)
  (define-fun t0 () (_ BitVec 32)
    #x00000000)
  (define-fun ebx () (_ BitVec 32)
    #x21524111)
)
```

http://rise4fun.com/Z3/j2Y

Генерация полезной нагрузки

- Data Execution Prevention (DEP)
- Return Oriented Programming
  - сначала весело, потом больно:
    - сотни и тысячи ROP-гаджетов
    - «плохие символы»
    - найти подходящий гажджет может быть сложно
- Усилия исследователей направлены на решение задачи автоматической генерации ROP-цепочек

Генерация полезной нагрузки



The (In) Complete Guide To Code Analysis Carpentry

Интересный пример от 0vercl0k'a

- **Дано:**
  - можем писать в eax, но только печатные ASCII символы
  - имеем ROP-гаджеты вида:
    - add eax, 0xc9f4458b; add eax, 0xdeadbeef;
    - add eax, 0x0fcf; add eax, 0x13b2;
    - add eax, 0x1337; add eax, 0x42;
- **Задача:**
  - сделать так, чтобы eax = 0xb00bdead
    - определить начальное значение eax
    - найти минимальную последовательность вызовов гаджетов

https://twitter.com/0vercl0k/status/359094667729580032

Интересный пример от 0vercl0k'a

- **Решение:** http://rise4fun.com/Z3Py/OrzP
  - assert(init_eax + g1*0xc9f4458b +
    g2*0xdeadbeef +
    g3*0x0fcf +
    g4*0x13b2 +
    g5*0x1337 +
    g6*0x42 = 0xb00bdead)
  - assert (ascii_printable(init_eax))
  - sum (g1,g2,g3,g4,g5,g6) --> min
- **Ответ:**
  - 0x522e707c + 3*0xc9f4458b + 8*0x13b2 = 0xb00bdead

Ссылки по теме

- Fun uses for an SMT solver
- AEG: Automatic Exploit Generation on Source Code
- Symbolic Exploit Assistant
- Transformation-aware Exploit Generation using a HI-CFG
- Hanging on a ROPe
- Code analysis carpentry
- Q: Exploit Hardening Made Easy
- A framework for automated architecture-independent gadget search
- Twitter 0vercl0ck
- OptiROP: the art of hunting ROP gadgets

Анализ защиты и SMT

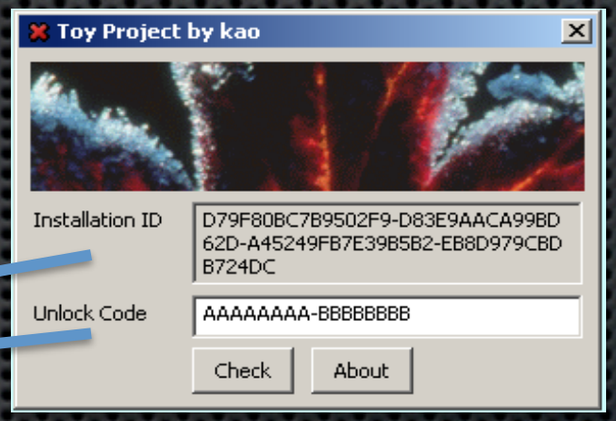
- Автоматическое создание кейгенов
  - решение Рольфа Роллеса (Rolf Rolles) крякми Kao's Toy Project
    - для решения Рольф использует свои инструменты, к сожалению закрытые
  - детальный анализ от Dcoder

Kao's Toy Project

```
; Attributes: bp-based frame

checkUnlockCode proc near

InstallationID= byte ptr -21h
UnlockCodePart1= dword ptr  8
UnlockCodePart2= dword ptr  0Ch

push    ebp
mov     ebp, esp
add     esp, 0FFFFFFDCh
mov     ecx, 32         ; count
mov     esi, offset unk_4093A8
lea     edi, [ebp+InstallationID] ; ciphertext
mov     edx, [ebp+UnlockCodePart1] ; key_part1
mov     ebx, [ebp+UnlockCodePart2] ; key_part2
```

```
decrypt_401105:
lodsb
sub     al, bl
xor     al, dl
stosb
rol     edx, 1
rol     ebx, 1
loop    decrypt_401105
```

```
loc_401111:
mov     byte ptr [edi], 0
push    offset PlainText ; "0how4zdy81jpe5xfu92kar6cgiq3lst7"
lea     eax, [ebp+InstallationID]
push    eax             ; lpDecryptedInstallationID
call    lstrcmpA
leave
retn    8
checkUnlockCode endp
```




```
int checkUnlockCode(u8 InstallationID[32], u32 UnlockCode1, u32 UnlockCode2)
{
    u8 PlainText = "0how4zdy81jpe5xfu92kar6cgiq3lst7";
    u8 DecryptedInstallationID[32];

    u8 i;
    for( i=0; i < 32; ++i)
    {
        DecryptedInstallationID[i] = (InstallationID[i] - UnlockCode1) ^ UnlockCode2;

        UnlockCode1 = ROL(UnlockCode1, 1);
        UnlockCode2 = ROL(UnlockCode2, 1);
    }

    return lstrcmp(PlainText, DecryptedInstallationID);
}
```

Инструменты

- IDA Pro
- Binary Analysis Platform (BAP)
  - набор утилит и API для анализа бинарного кода
  - перевод в Intermediate Language (IL, IR)
  - преобразование IL представления в SMT-формулу
  - http://bap.ece.cmu.edu/
- SMT-решатель z3

Зачем нужен IL?

- Увеличивает точность анализа
  - раскрывает неявные эффекты при выполнении команд языка ассемблера
- Упрощение и оптимизация
- Формализация позволяет кратко и точно описать анализ, и затем перейти к простой реализации

Делай раз !

1. Переводим алгоритм проверки в IL
   - ./toil -binrange ~/toyproject.exe 0x401105 0x401111 -o checkUnlockCode.il

Результат перевода в IL

- decrypt:

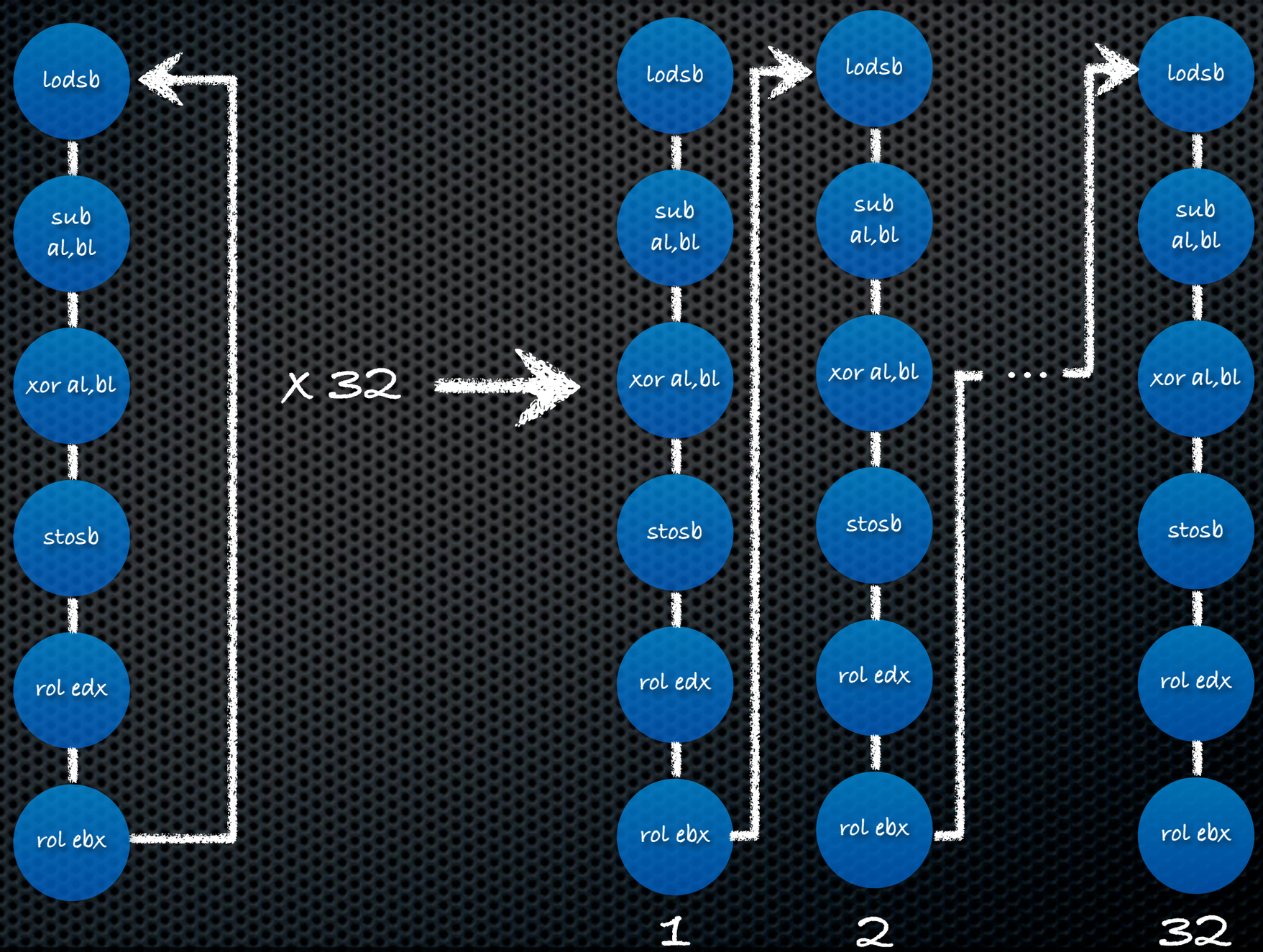
```
lodsb
sub     al, bl
xor     al, dl
stosb
rol     edx, 1
rol     ebx, 1
loop    decrypt
```

```
addr 0x401106 @asm "sub    %bl,%al"
label pc_0x401106
T_t:u8 = low:u8(R_EAX:u32)
R_EAX:u32 =
  concat:[extract:31:8:[R_EAX:u32]][low:u8(R_EAX:u32) - low:u8(R_EBX:u32)]
R_CF:bool = T_t:u8 < low:u8(R_EBX:u32)
R_OF:bool =
  high:bool((T_t:u8 ^ low:u8(R_EBX:u32)) & (T_t:u8 ^ low:u8(R_EAX:u32)))
R_AF:bool =
  0x10:u8 == (0x10:u8 & (low:u8(R_EAX:u32) ^ T_t:u8 ^ low:u8(R_EBX:u32)))
R_PF:bool =
  ~low:bool(low:u8(R_EAX:u32) >> 7:u8 ^ low:u8(R_EAX:u32) >> 6:u8 ^
            low:u8(R_EAX:u32) >> 5:u8 ^ low:u8(R_EAX:u32) >> 4:u8 ^
            low:u8(R_EAX:u32) >> 3:u8 ^ low:u8(R_EAX:u32) >> 2:u8 ^
            low:u8(R_EAX:u32) >> 1:u8 ^ low:u8(R_EAX:u32))
R_SF:bool = high:bool(low:u8(R_EAX:u32))
R_ZF:bool = 0:u8 == low:u8(R_EAX:u32)
addr 0x401108 @asm "xor    %dl,%al"
label pc_0x401108
R_EAX:u32 =
  concat:[extract:31:8:[R_EAX:u32]][low:u8(R_EAX:u32) ^ low:u8(R_EDX:u32)]
R_OF:bool = false
R_CF:bool = false
R_AF:bool = unknown "AF is undefined after xor":bool
R_PF:bool =
  ~low:bool(low:u8(R_EAX:u32) >> 7:u8 ^ low:u8(R_EAX:u32) >> 6:u8 ^
            low:u8(R_EAX:u32) >> 5:u8 ^ low:u8(R_EAX:u32) >> 4:u8 ^
            low:u8(R_EAX:u32) >> 3:u8 ^ low:u8(R_EAX:u32) >> 2:u8 ^
            low:u8(R_EAX:u32) >> 1:u8 ^ low:u8(R_EAX:u32))
R_SF:bool = high:bool(low:u8(R_EAX:u32))
R_ZF:bool = 0:u8 == low:u8(R_EAX:u32)
addr 0x40110a @asm "stos   %al,%es:(%edi)"
label pc_0x40110a
mem:?u32 = mem:?u32 with [R_EDI:u32, e_little]:u8 = low:u8(R_EAX:u32)
R_EDI:u32 = R_EDI:u32 + R_DFLAG:u32
addr 0x40110b @asm "rol    %edx"
label pc_0x40110b
T_origCOUNT:u32 = 1:u32
R_EDX:u32 =
  R_EDX:u32 << T_origCOUNT:u32 | R_EDX:u32 >> 0x20:u32 - T_origCOUNT:u32
R_CF:bool =
  if T_origCOUNT:u32 == 0:u32 then R_CF:bool else low:bool(R_EDX:u32)
R_OF:bool =
  if T_origCOUNT:u32 == 0:u32 then R_OF:bool else
  if T_origCOUNT:u32 == 1:u32 then R_CF:bool ^ high:bool(R_EDX:u32) else
  unknown "OF undefined after rotate of more then 1 bit":bool
addr 0x40110d @asm "rol    %ebx"
label pc_0x40110d
T_origCOUNT_84:u32 = 1:u32
R_EBX:u32 =
```

Делай два !

1. Переводим алгоритм проверки в IL
   - ./toil -binrange ~/toyproject.exe 0x401105 0x401111 -o checkUnlockCode.il
2. Трансформируем IL в SSA-форму, раскручиваем циклы
   - ./iltrans -il checkUnlockCode.il -to-ssa -simp-ssa -to-cfg -unroll 31 -rm-cycles \ -rm-indirect-ast -to-ast -normalize-mem -flatten-mem -pp-ast \ checkUnlockUnroll.il
   - egrep -v '^cjmp*$' checkUnlockUnroll.il > checkUnlockUnrollOpt.il

Результат раскрутки цикла



Делай три !

1. Переводим алгоритм проверки в IL
   - ./toil -binrange ~/toyproject.exe 0x401105 0x401111 -o checkUnlockCode.il
2. Трансформируем IL в SSA-форму, раскручиваем циклы
   - ./iltrans -il checkUnlockCode.il -to-ssa -simp-ssa -to-cfg -unroll 31 -rm-cycles \ -rm-indirect-ast -to-ast -normalize-mem -flatten-mem -pp-ast \ checkUnlockUnroll.il
   - egrep -v '^cjmp*$' checkUnlockUnroll.il > checkUnlockUnrollOpt.il
3. Создаем из IL SMT формулу
   - ./topredicate -il checkUnlockUnrollOpt.il -noopt -solver z3 -stp-out checkLoop.smt

Замечания

- ./topredicate - сделала основную тяжёлую работу, **НО**...
- т.к. утилита предназначена для другой задачи, нужно чуть-чуть модифицировать формулу checkLoop.smt:
  - в строке 18 заменить assert на строку
    define-fun alg () (Array (_ BitVec 32) (_ BitVec 8)
  - в строке 921 заменить false на ?mem_array_83_670
- подправленная формула http://rise4fun.com/Z3/cJ6h

Генерируем предусловие и постусловие

```
print '############ Preconditions and Postconditions ################'

mem_installation_id = \
'(assert (= (select mem_array_242_84 (_ bv%d 32)) (_ bv%d 8)))'
mem_output     = '(assert (= (select alg (_ bv%d 32)) (_ bv%d 8)))'

# формулируем предусловия
print '(assert (= R_ESI_2_86 (_ bv0 32)))'  # ESI = 0
print '(assert (= R_EDI_3_80 (_ bv32 32)))' # EDI =32
print '(assert (= R_DFLAG_16 (_ bv1 32)))'  # величина инкремента для ESI и EDI
print ''

# считываем Installation ID из памяти
buf = idaapi.dbg_read_memory(0x004093A8, 32)
for i in range(32):
    print mem_installation_id % (i, ord(buf[i]))
print ''
# формулируем постусловие
# считываем 0how4zdy81jpe5xfu92kar6cgiq3lst7 из памяти
buf = idaapi.dbg_read_memory(0x00409185, 32)
for i in range(32):
    print mem_output % (i+32, ord(buf[i]))
```

Тададада!

z3 Microsoft Research

Is this formula satisfiable?

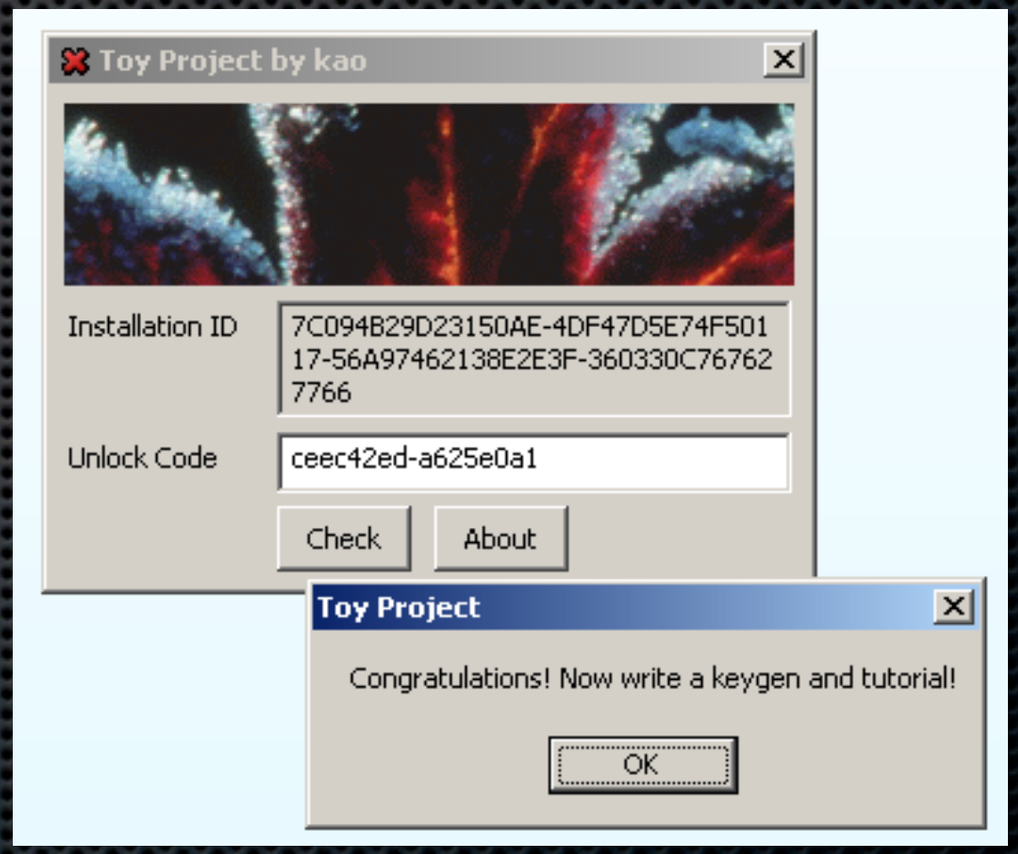
```
(set-logic QF_ABV)
(set-info :smt-lib-version 2.0)
(set-option :produce-assignments true)

; free variables:
(declare-fun R_DFLAG_16 () (_ BitVec 32))
(declare-fun R_EDI_3_80 () (_ BitVec 32))
(declare-fun R_EDX_8_81 () (_ BitVec 32))
(declare-fun R_EAX_5_82 () (_ BitVec 32))
(declare-fun mem_array_242_84 () (Array (_ BitVec 32) (_ BitVec 8)))
(declare-fun R_EBX_6_85 () (_ BitVec 32))
(declare-fun R_ESI_2_86 () (_ BitVec 32))
(declare-fun R_ECX_7_87 () (_ BitVec 32))
; end free variables.

(define-fun alg () (Array (_ BitVec 32) (_ BitVec 8))
 (let ((?R_EDI_3_0 R_EDI_3_80))
 (let ((?R_EDX_8_1 R_EDX_8_81))
 (let ((?R_EAX_5_2 R_EAX_5_82))
 (let ((?mem_array_83_3 mem_array_242_84))
 (let ((?R_EBX_6_4 R_EBX_6_85))
 (let ((?R_ESI_2_5 R_ESI_2_86))
```

tutorial

home video permalink

'►' shortcut: Alt+B

```
sat
(model
  (define-fun R_EBX_6_85 () (_ BitVec 32)
    #x68c9a24c)
  (define-fun mem_array_242_84 () (Array (_ BitVec 32) (_ BitVec 8))
    (_ as-array k!0))
  (define-fun R_EDX_8_81 () (_ BitVec 32)
    #xceec42ed)
  (define-fun REAL_EBX () (_ BitVec 32)
    #xa625e0a1)
```



http://rise4fun.com/Z3/6u7

DEMO

Ссылки по теме

- SMT Solvers for Software Security
- Microsoft Security Science. SMT Solvers for Software Security
- Extending SAT Solvers to Cryptographic Problems
- Applications of SAT Solvers to Cryptanalysis of Hash Functions
- The Case for Semantics-Based Methods in Reverse Engineering

Спасибо за внимание!