UNCLASSIFIED

Australian Government
Department of Defence
Science and Technology

Analyzing trigger-based malware with S2E

Adrian Herrera
0xadr1an

Defence Science and Technology Group

April 25, 2023

$ whoami

- Researcher with the Defence Science and Technology (DST) Group
- PhD student at the Australian National University (ANU)
- S2E developer/maintainer

Outline

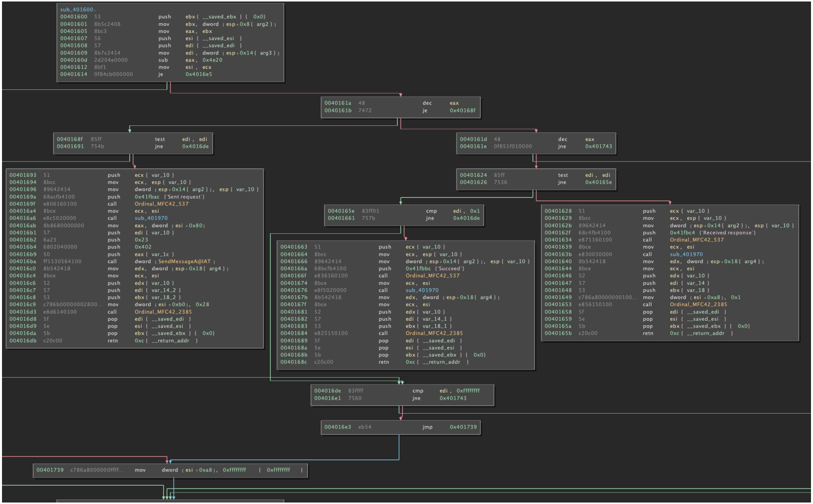
1. Symbolic execution
2. S2E
3. Trigger-based malware
4. Analyzing trigger-based malware with S2E

Symbolic execution

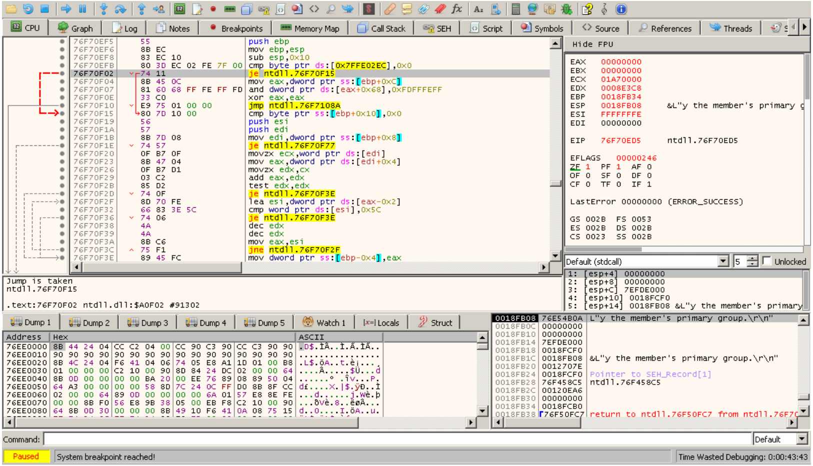
Introduction

What are typical approaches to reversing malware?

Introduction



Introduction



Introduction

Can we get the best of both worlds?

Symbolic execution

Program analysis technique for **systematically** exploring **all** paths through a program*

Symbolic execution

Program analysis technique for **systematically** exploring **all** paths through a program*

*Conditions apply

Symbolic execution

- Program input is provided as a **symbolic** value rather than **concrete** data
- Operations (e.g., addition, assignment, etc.) are performed on these symbolic values to generate **symbolic expressions**
- Conditional statements result in an execution **fork**
- A **constraint solver** is invoked to find a solution to the symbolic expressions (if one exists) and generates a concrete input for the path explored

An example[1]

```
void foobar(int a, int b) {
    int x = 1, y = 0;
    if (a != 0) {
        y = 3 + x;
        if (b == 0) {
            x = 2 * (a + b);
        }
    }

    assert(x - y != 0);
}
```

[1] "A Survey of Symbolic Execution Techniques", R. Baldoni *et al.*

An example

```
//   a ↦ α, b ↦ β
void foobar(int a, int b) {
    int x = 1, y = 0;
    if (a != 0) {
        y = 3 + x;
        if (b == 0) {
            x = 2 * (a + b);
        }
    }

    assert(x - y != 0);
}
```

An example

```
void foobar(int a, int b) {
    //  a ↦ α, b ↦ β, x ↦ 1, y ↦ 0
    int x = 1, y = 0;
    if (a != 0) {
        y = 3 + x;
        if (b == 0) {
            x = 2 * (a + b);
        }
    }

    assert(x - y != 0);
}
```

An example

```
void foobar(int a, int b) {
    int x = 1, y = 0;
    // Two possible execution paths:
    //  1.  a ↦ ¬(α ≠ 0), b ↦ β, x ↦ 1, y ↦ 0
    //  2.  a ↦ α ≠ 0, b ↦ β, x ↦ 1, y ↦ 0
    if (a != 0) {
        y = 3 + x;
        if (b == 0) {
            x = 2 * (a + b);
        }
    }

    assert(x - y != 0);
}
```

An example

```
void foobar(int a, int b) {
    int x = 1, y = 0;
    if (a != 0) {
        y = 3 + x;
        if (b == 0) {
            x = 2 * (a + b);
        }
    }

    // Path 1
    //   a ↦ ¬(α ≠ 0), b ↦ β, x ↦ 1, y ↦ 0
    //   1 − 0 = 1 ≠ 0
    assert(x - y != 0);
}
```

An example

```
void foobar(int a, int b) {
    int x = 1, y = 0;
    if (a != 0) {
        // Path 2
        //   a ↦ α ≠ 0, b ↦ β, x ↦ 1, y ↦ 3 + 1 = 4
        y = 3 + x;
        if (b == 0) {
            x = 2 * (a + b);
        }
    }

    assert(x - y != 0);
}
```

An example

```
void foobar(int a, int b) {
    int x = 1, y = 0;
    if (a != 0) {
        y = 3 + x;
        // Two possible execution paths:
        //   3.   a ↦ α ≠ 0, b ↦ ¬(β = 0), x ↦ 1, y ↦ 4
        //   4.   a ↦ α ≠ 0, b ↦ β = 0, x ↦ 1, y ↦ 4
        if (b == 0) {
            x = 2 * (a + b);
        }
    }

    assert(x - y != 0);
}
```

An example

```
void foobar(int a, int b) {
    int x = 1, y = 0;
    if (a != 0) {
        y = 3 + x;
        if (b == 0) {
            x = 2 * (a + b);
        }
    }

    // Path 3
    //   a ↦ α ≠ 0, b ↦ ¬(β = 0), x ↦ 1, y ↦ 4
    //   1 − 4 = −3 ≠ 0
    assert(x - y != 0);
}
```

An example

```
void foobar(int a, int b) {
    int x = 1, y = 0;
    if (a != 0) {
        y = 3 + x;
        if (b == 0) {
            // Path 4
            //   a ↦ α ≠ 0, b ↦ β = 0,
            //   x ↦ 2 × [(α ≠ 0) + (β = 0)], y ↦ 4
            x = 2 * (a + b);
        }
    }

    assert(x - y != 0);
}
```

An example

```
void foobar(int a, int b) {
    int x = 1, y = 0;
    if (a != 0) {
        y = 3 + x;
        if (b == 0) {
            x = 2 * (a + b);
        }
    }

    //  a ↦ α ≠ 0, b ↦ β = 0,
    //  x ↦ 2 × [(α ≠ 0) + (β = 0)], y ↦ 4
    assert(x - y != 0);
}
```

An example

```
void foobar(int a, int b) {
    int x = 1, y = 0;
    if (a != 0) {
        y = 3 + x;
        if (b == 0) {
            x = 2 * (a + b);
        }
    }

    //  2 × [(α ≠ 0) + (β = 0)] − 4 = 0
    //  a ↦ 2, b ↦ 0
    assert(x - y != 0);
}
```

An example

```
void foobar(int a, int b) {
    int x = 1, y = 0;
    if (a != 0) {
        y = 3 + x;
        if (b == 0) {
            x = 2 * (a + b);
        }
    }

    assert(x - y != 0);
}

// All paths (×4) explored
```

S2E

Available tools

Many symbolic execution engines available

Available tools

Many symbolic execution engines available

MANTICORE

TRITON
Dynamic Binary Analysis

KLEE

S2E

Available tools

Many symbolic execution engines available

MANTICORE

TRITON
Dynamic Binary Analysis

S2E

S2E introduction

S2E is a platform for in-vivo multi-path analysis of software systems

S2E introduction

S2E is a platform for in-vivo multi-path analysis of software systems

- Extensible
- Write your own tools

S2E introduction

S2E is a platform for in-vivo multi-path analysis of software systems

- On real OSes, with real apps, libraries, drivers

S2E introduction

S2E is a platform for in-vivo multi-path analysis of software systems

- Symbolic execution
- Concolic execution
- State merging
- Fuzzing
- ...

S2E introduction

S2E is a platform for in-vivo multi-path analysis of software systems

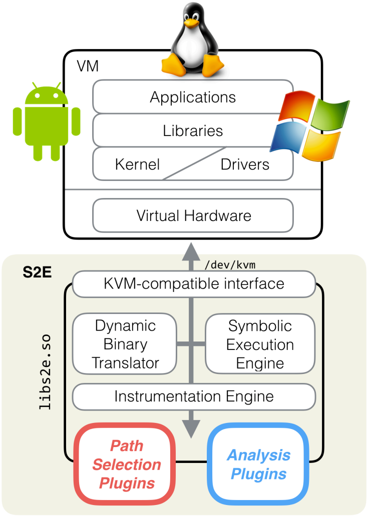
- Bug finding
- Verification
- Testing
- Security checking

S2E introduction

S2E is a platform for in-vivo multi-path analysis of software systems

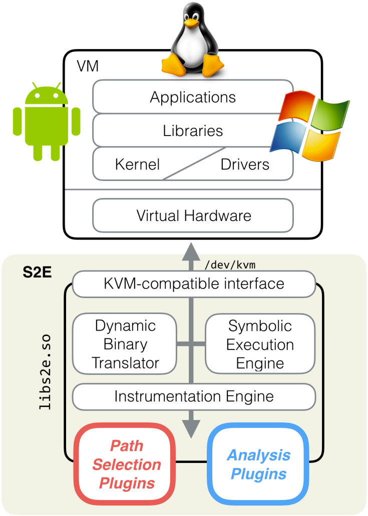
- Pretty much anything that runs on a computer

S2E architecture



- S2E uses **QEMU**
- S2E intercepts and replaces `/dev/kvm`
- QEMU's dynamic binary translator translates guest instructions to **LLVM**
- LLVM instructions symbolically executed by **KLEE**

S2E architecture



Path selection plugins

- What input to make symbolic?
- What input to make concrete?
- Search heuristics

Analysis plugins

- Check for crashes
- Check for vulnerability conditions
- Performance measurements

Why S2E?

- Works on unmodified binaries
- Operates at any level of the software stack
- Does not require environment modelling

Why S2E?

- Works on unmodified binaries
- Operates at any level of the software stack
- Does not require environment modelling

Perfect for malware analysis

Trigger-based malware

Trigger-based malware

"Hidden behavior/certain code paths that are only executed under certain *trigger conditions*" [2]

[2] "Automatically Identifying Trigger-based Behavior in Malware", D. Brumley *et al.*

Trigger examples

- Internet connectivity
- Mutex objects
- Existence of files
- Existence of Registry entries
- Data read from a file
- ...

Trigger example – time [3]

```
SYSTEMTIME systime;
LPCSTR site = "https://federation.edu.au/icsl/mre2019";

GetLocalTime(&systime);

if (9 == systime.wDay) {
    if (10 == systime.wHour) {
        if (11 == systime.wMonth) {
            if (6 == systime.wMinute) {
                ddos(site);
            }
        }
    }
}
```

[3] "Automatically Identifying Trigger-based Behavior in Malware", D. Brumley *et al.*

Trigger example – network

```
mov     esi, data_4313d0  {"http://www.iuqerfsodp9ifjaposdfj…"}
lea     edi, [esp+0x8 {var_50}]
xor     eax, eax  {0x0}
rep movsd dword [edi], [esi]  {0x0}
movsb   byte [edi], [esi]  {0x0}
mov     dword [esp+0x41 {var_17}], eax  {0x0}
mov     dword [esp+0x45 {var_13}], eax  {0x0}
mov     dword [esp+0x49 {var_f}], eax  {0x0}
mov     dword [esp+0x4d {var_b}], eax  {0x0}
mov     dword [esp+0x51 {var_7}], eax  {0x0}
mov     word [esp+0x55 {var_3}], ax  {0x0}
push    eax  {0x0}
push    eax {var_60}  {0x0}
push    eax {var_64}  {0x0}
push    0x1 {var_68}
push    eax  {0x0}
mov     byte [esp+0x6b {var_1}], al  {0x0}
call    dword [InternetOpenA@IAT]
push    0x0
push    0x84000000 {var_60}  {0x84000000}
push    0x0 {var_64}
lea     ecx, [esp+0x14 {var_50}]
mov     esi, eax
push    0x0 {var_68}
push    ecx {var_50} {var_6c}
push    esi {var_70}
call    dword [InternetOpenUrlA@IAT]
mov     edi, eax
push    esi {var_5c}
mov     esi, dword [InternetCloseHandle@IAT]
test    edi, edi
jne     0x4081bc
```

```
call    esi
push    edi {var_5c_1}
call    esi
pop     edi {__saved_edi}
```

```
call    esi
push    0x0
call    esi
call    sub_408090
```

Analyzing trigger-based malware

Why is it hard?

Analyzing trigger-based malware

Why is it hard?

- Typical **dynamic analysis** cannot determine the trigger **conditions** to go down the correct path
- Code may be **obfuscated**, so hard to determine trigger conditions **statically**

Analyzing trigger-based malware

Why is it hard?

- Typical **dynamic analysis** cannot determine the trigger **conditions** to go down the correct path
- Code may be **obfuscated**, so hard to determine trigger conditions **statically**

Symbolic execution can help

Analyzing trigger-based malware with S2E

Why not fuzz?

Possible approach:

1. Identify trigger types of interest (e.g., time, network, etc.)
2. Generate random trigger inputs
3. `goto` 2 until trigger condition is met

Why not fuzz?

Possible approach:

1. Identify trigger types of interest (e.g., time, network, etc.)
2. Generate random trigger inputs
3. `goto` 2 until trigger condition is met

Problems:

- Highly **inefficient** – small probability of guessing the *exact* trigger value
- Not interested in **exploring** program – only in the trigger path

Symbolic execution approach

1. Identify trigger types of interest (e.g., time, network, etc.)
2. Represent trigger inputs symbolically
3. Collect constraints and fork at conditional statements
4. Solve constraints $\rightarrow$ trigger values

S2E approach

1. Hook trigger sources (e.g., `GetLocalTime`, `InternetOpenURL`, etc.)
2. Make return value symbolic (via S2E API)

S2E approach

1. Hook trigger sources (e.g., `GetLocalTime`, `InternetOpenURL`, etc.)
2. Make return value symbolic (via S2E API)

S2E handles everything else

S2E approach

1. Hook trigger sources (e.g., `GetLocalTime`, `InternetOpenURL`, etc.)
2. Make return value symbolic (via S2E API)

S2E handles everything else

Hook with **EasyHook** (`https://easyhook.github.io/`)

S2E example – time

```
SYSTEMTIME systime;
LPCSTR site = "https://federation.edu.au/icsl/mre2019";

GetLocalTime(&systime);

if (9 == systime.wDay) {
    if (10 == systime.wHour) {
        if (11 == systime.wMonth) {
            if (6 == systime.wMinute) {
                ddos(site);
            }
        }
    }
}
```

S2E example – time

```
#include <s2e/s2e.h>

static void WINAPI GetLocalTimeHook(
        LPSYSTEMTIME lpSystemTime) {
    // Get concrete value
    GetLocalTime(lpSytemTime);

    // Make symbolic
    S2EMakeSymbolic(lpSystemTime,
                    sizeof(*lpSystemTime),
                    "systime");
}

// TODO: Initialize EasyHook
```

S2E example – time

S2E example – time

S2E produces the following trigger input:

```
v0_systime_0 = {0x0, 0x0, /* wYear */
                0xb, 0x0, /* wMonth */
                0x0, 0x0, /* wDayOfWeek */
                0x9, 0x0, /* wDay */
                0xa, 0x0, /* wHour */
                0x6, 0x0, /* wMinute */
                0x0, 0x0, /* wSecond */
                0x0, 0x0} /* wMilliseconds */
```

This is a byte-level representation of expected constraints:

$$systime.wDay = 9 \wedge \ systime.wHour = 10$$
$$\wedge \ systime.wMonth = 11 \wedge \ systime.wMinute = 6$$

S2E example – WannaCry

```
mov     esi, data_4313d0  {"http://www.iuqerfsodp9ifjaposdfj…"}
lea     edi, [esp+0x8 {var_50}]
xor     eax, eax  {0x0}
rep movsd dword [edi], [esi]  {0x0}
movsb   byte [edi], [esi]  {0x0}
mov     dword [esp+0x41 {var_17}], eax  {0x0}
mov     dword [esp+0x45 {var_13}], eax  {0x0}
mov     dword [esp+0x49 {var_f}], eax  {0x0}
mov     dword [esp+0x4d {var_b}], eax  {0x0}
mov     dword [esp+0x51 {var_7}], eax  {0x0}
mov     word [esp+0x55 {var_3}], ax  {0x0}
push    eax  {0x0}
push    eax {var_60}  {0x0}
push    eax {var_64}  {0x0}
push    0x1 {var_68}
push    eax  {0x0}
mov     byte [esp+0x6b {var_1}], al  {0x0}
call    dword [InternetOpenA@IAT]
push    0x0
push    0x84000000 {var_60}  {0x84000000}
push    0x0 {var_64}
lea     ecx, [esp+0x14 {var_50}]
mov     esi, eax
push    0x0 {var_68}
push    ecx {var_50} {var_6c}
push    esi {var_70}
call    dword [InternetOpenUrlA@IAT]
mov     edi, eax
push    esi {var_5c}
mov     esi, dword [InternetCloseHandle@IAT]
test    edi, edi
jne     0x4081bc
```

```
call    esi
push    edi {var_5c_1}
call    esi
pop     edi {__saved_edi}
```

```
call    esi
push    0x0
call    esi
call    sub_408090
```

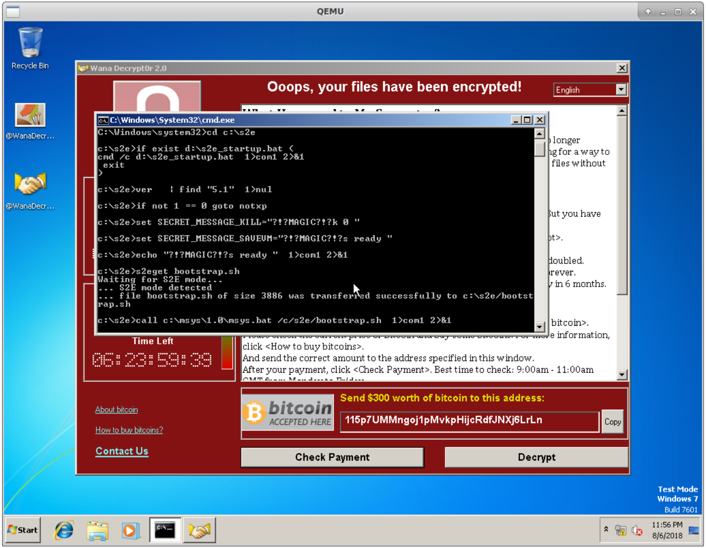
S2E example – WannaCry

```
static std::set<HINTERNET> dummyHandles;

static HINTERNET WINAPI InternetOpenUrlAHook(
        HINTERNET hInternet, /* ... */ ) {
    UINT8 returnResource = S2ESymbolicChar("hInternet", 1);
    if (returnResource) {
        // Create and return a dummy handle
        HINTERNET resourceHandle = (HINTERNET) malloc(
            sizeof(HINTERNET));
        dummyHandles.insert(resourceHandle);
        return resourceHandle;
    } else {
        // Simulate InternetOpenUrlA "failing"
        return NULL;
    }
}
```

S2E example – WannaCry

```
static BOOL WINAPI InternetCloseHandleHook(
        HINTERNET hInternet) {
    std::set<HINTERNET>::iterator it =
        dummyHandles.find(hInternet);

    if (it == dummyHandles.end()) {
        // Could be real a real handle
        return InternetCloseHandle(hInternet);
    } else {
        // A dummy handle
        free(*it);
        dummyHandles.erase(it);
        return TRUE;
    }
}
```

S2E example – WannaCry



Conclusion

- Recreated David Brumley's paper in S2E
- Explore **more** of the program than a typical dynamic analysis
- **Scalability** is an issue

All material available at

`https://github.com/adrianherrera/malware-s2e`

Conclusion

- Recreated David Brumley's paper in S2E
- Explore **more** of the program than a typical dynamic analysis
- **Scalability** is an issue

All material available at

`https://github.com/adrianherrera/malware-s2e`

Questions?