

UNCLASSIFIED



Australian Government

Department of Defence

Science and Technology

Analyzing trigger-based malware with S2E

Adrian Herrera

 0xad1an

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

The screenshot displays a debugger interface with the following components:

- Assembly View:** Shows assembly instructions from address 76F70E05 to 76F70F3E. Key instructions include:
 - `push ebp`
 - `mov ebp, esp`
 - `sub esp, 0x10`
 - `cmp byte ptr ds:[0x7FFE02EC], 0x0`
 - `je ntdll!.76F70F15`
 - `mov eax, dword ptr ss:[ebp+0xC]`
 - `and dword ptr ds:[eax+0x68], 0xFDFDFDFDF`
 - `xor eax, eax`
 - `jmp ntdll!.76F71084`
 - `cmp byte ptr ss:[ebp+0x10], 0x0`
 - `push esi`
 - `push edi`
 - `mov edi, dword ptr ss:[ebp+0x8]`
 - `je ntdll!.76F70F27`
 - `movzx ecx, word ptr ds:[edi]`
 - `mov eax, dword ptr ds:[edi+0x4]`
 - `movzx edx, cx`
 - `add eax, edx`
 - `test edx, edx`
 - `je ntdll!.76F70F3E`
 - `lea esi, dword ptr ds:[eax-0x2]`
 - `cmp word ptr ds:[esi], 0x5C`
 - `je ntdll!.76F70F3E`
 - `dec ecx`
 - `dec edx`
 - `mov eax, esi`
 - `je ntdll!.76F70F3E`
 - `mov dword ptr ss:[ebp-0x4], eax`
- Registers View:** Shows register values:
 - EAX: 00000000
 - EBX: 00000000
 - ECX: 01A70000
 - EDX: 0008E3C8
 - EBP: 0018FB34
 - ESP: 0018FB08
 - ESI: FFFFFFFE
 - EDI: 00000000
 - EIP: 76F70E05
- System Information:**
 - EFlags: 00000246
 - ZE 1 PF 1 AF 0
 - OF 0 SF 0 DF 0
 - CF 0 TF 0 IF 1
 - LastError: 00000000 (ERROR_SUCCESS)
 - GS 0028 FS 0053
 - ES 0028 DS 0028
 - CS 0023 SS 0028
- Memory Dump:** Shows a hex dump of memory starting at address 76E00000. The dump contains ASCII text including:
 - `..ds.I.A..I.A.I.A..`
 - `..S.S.S.S.S.S.S.S..`
 - `..A...S.U.D..`
 - `..*..V..P..`
 - `..X..S.Y..I..`
 - `..D.V.S..e.A..`
 - `..D...I..O.A..U..`
- Command Line:** Shows the command: `Paused System breakpoint reached`

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 example¹

```
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.  $a \mapsto \neg(\alpha \neq 0), b \mapsto \beta, x \mapsto 1, y \mapsto 0$ 
    // 2.  $a \mapsto \alpha \neq 0, b \mapsto \beta, x \mapsto 1, y \mapsto 0$ 
    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  
    //  $a \mapsto \neg(\alpha \neq 0), b \mapsto \beta, x \mapsto 1, y \mapsto 0$   
    //  $1 - 0 = 1 \neq 0$   
    assert(x - y != 0);  
}
```


An example

```
void foobar(int a, int b) {  
    int x = 1, y = 0;  
    if (a != 0) {  
        // Path 2  
        //  $a \mapsto \alpha \neq 0, b \mapsto \beta, x \mapsto 1, y \mapsto 3 + 1 = 4$   
        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.  $a \mapsto \alpha \neq 0, b \mapsto \neg(\beta = 0), x \mapsto 1, y \mapsto 4$ 
        // 4.  $a \mapsto \alpha \neq 0, b \mapsto \beta = 0, x \mapsto 1, y \mapsto 4$ 
        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 ↦  $\alpha \neq 0$ , b ↦  $\neg(\beta = 0)$ , x ↦ 1, y ↦ 4
    // 1 - 4 = -3  $\neq$  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  
            //  $a \mapsto \alpha \neq 0, b \mapsto \beta = 0,$   
            //  $x \mapsto 2 \times [(\alpha \neq 0) + (\beta = 0)], y \mapsto 4$   
            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 ↦  $\alpha \neq 0$ , b ↦  $\beta = 0$ ,  
    // x ↦  $2 \times [(\alpha \neq 0) + (\beta = 0)]$ , 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);  
        }  
    }  
  
    //  $2 \times [(a \neq 0) + (b = 0)] - 4 = 0$   
    //  $a \mapsto 2, b \mapsto 0$   
    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



TRILON
Dynamic Binary Analysis



S2E

Available tools

Many symbolic execution engines available



TRILON
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

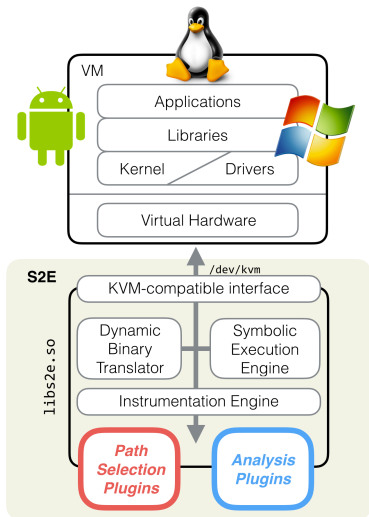
- Bug finding
- Verification
- Testing
- Security checking

S2E introduction

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

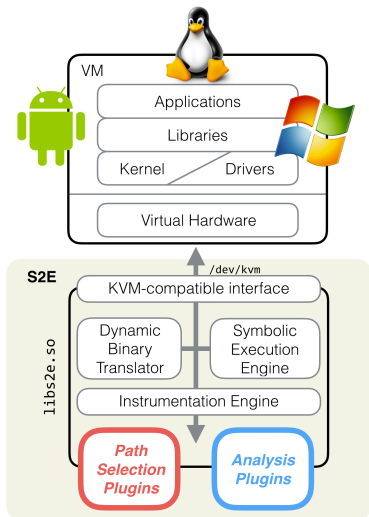
- Pretty much anything that runs on a computer

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(lpSystemTime);

    // 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:

$$\begin{aligned} & \text{*systime.wDay* = 9} \wedge \text{*systime.wHour* = 10} \\ & \wedge \text{*systime.wMonth* = 11} \wedge \text{*systime.wMinute* = 6} \end{aligned}$$

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

The screenshot shows a QEMU virtual machine running a Windows 7 desktop. A ransomware window titled "Wana Decrypt0r 2.0" is the central focus. The window has a red header with the text "Oops, your files have been encrypted!". Below the header, there is a Bitcoin logo and the text "Send \$300 worth of Bitcoin to this address:". A Bitcoin address is displayed in a red box: `115p7UMMngo1pMvvpHjicRdfJNXj6LrLn`. There are two buttons at the bottom: "Check Payment" and "Decrypt".

A command prompt window is open over the ransomware window, showing the following commands and output:

```

C:\Windows\System32\cmd.exe
C:\Windows\system32>cd c:\s2e
c:\s2e>if exist d:\s2e_startup.bat (
cmd /c d:\s2e_startup.bat 1>conl 2>&1
exit
)
c:\s2e>ver | find "5.1" 1>nul
c:\s2e>if not 1 == 0 goto notxp
c:\s2e>set SECRET_MESSAGE_KILL="?!?MAGIC?!?k 0 "
c:\s2e>set SECRET_MESSAGE_SAEUHM="?!?MAGIC?!?s ready "
c:\s2e>echo "?!?MAGIC?!?s ready " 1>conl 2>&1
c:\s2e>s2eget bootstrap.sh
Waiting for S2E mode...
... S2E mode detected
... file bootstrap.sh of size 3886 was transferred successfully to c:\s2e/boots
rap.sh
c:\s2e>call c:\nsys\1.0\nsys.bat /c/s2e/bootstrap.sh 1>conl 2>&1
  
```

At the bottom of the ransomware window, there is a "Time Left" section showing a timer at `06:23:59:39`. Below the timer, there are links for "About bitcoin", "How to buy bitcoins?", and "Contact Us".

The taskbar at the bottom of the desktop shows the Start button, several application icons, and the system tray with the date and time: 11:56 PM 8/6/2018. The system information in the bottom right corner indicates "Test Mode Windows 7 Build 7601".

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?