The S2E Platform

Testing Windows kernel drivers with symbolic execution

Vitaly Chipounov

https://s2e.systems

Ready-for-use docker image, demos, tutorials, source code, documentation

Dealing with Path Explosion

- Identifying bottlenecks
  Fork profiler, code coverage, perf top, etc.

- Using concrete inputs to guide symbolic execution
  Seed inputs from fuzzer + concolic execution

- How to deal with (very) large input files
  Selecting which bytes should be symbolic

- Parallelizing symbolic execution
  Running S2E on multiple cores

- Function annotations
  Symbolic fault injection

Testing Drivers

- Static analysis
  Microsoft PREfast, Static Driver Verifier

- Dynamic analysis
  Run the driver with different inputs
  Driver Verifier => stresses the driver by injecting faults

fastfat.sys

- FAT file system driver that ships with Windows
- Source code is available in the Windows Driver Kit
  Along with the source for many other drivers
- 31 KLOC, imports 240 functions from the kernel
  Hard to run in isolation (e.g., in KLEE)
- Input files are large
  Floppy image is 1.4MB large, FAT32 volumes even bigger

Building the Driver

```
ssh -CX userXX@issisp.anu.edu.au

$ export VS_PLATFORM=Win32
$ export VS_CONFIG=Debug
$ export REMOTE_HOST=issisp-s2e-win10
$ export REMOTE_FOLDER=$USER

$ ~/s2e/source/s2e/build-scripts/windows/remote-
msbuild.sh ~/s2e/source/windows-driver-samples/
filesys/fastfat/
```

Building the Driver

```
$ ls ~/s2e/source/windows-driver-samples/filesys/
fastfat/Debug/fastfat*

fastfat2.cer
fastfat2.inf
fastfat2.pdb
fastfat2.sys
fastfat2.sys.lines
```

It’s called fastfat2.sys to not conflict with the one that ships with the system

Creating a Driver Project

```
$ source s2e/s2e_activate
$ s2e new_project ~/s2e/source/windows-driver-
samples/filesys/fastfat/Debug/fastfat2.inf @@
```

- Creates a Windows XP project
  We'll use XP for the tutorial because of its low resource requirements (256MB of guest RAM)

- Windows 7, 8.1, and 10 are supported too
  Must recompile the driver in 64-bit mode
  VS_PLATFORM=x64

Running the Driver

- The project creation tool does its best effort to detect what you want to analyze and create the most appropriate configuration

- You still need to tell S2E how to actually start the driver and how to feed input files to it

Customizing bootstrap.sh

```
$ nano ~/s2e/projects/fastfat2/bootstrap.sh
```

- Initialize S2E monitoring driver (s2e.sys)
- Fetch files of the binary to test
  fastfat2.sys + fastfat2.inf
- Create a symbolic input file in a ramdisk
- Load the binary
  Requires manual configuration

$ nano ~/s2e/projects/fastfat2/bootstrap.sh

```
function execute_target {
    local TARGET
    local SYMB_FILE

    TARGET="$1"
    SYMB_FILE="$2"

    # Activate fault injection right before loading the driver
    ./drvctl.exe set_config FaultInjectionActive 0

    # Ask windows to load the driver
    install_driver "$(win_path "$TARGET")"

    sc start fastfat2

    imdisk -a -o hd -f x:\\input -m F:
    mount -a
}
```

If you already have a test suite for your binary, this is were you would call it

Initial Run and Coverage

```
$ cd s2e/projects/fastfat2
$ ./launch-s2e.sh

… wait for a few states to fork…

$ killall -9 qemu-system-i386



$ s2e coverage --sympath ~/s2e/source/windows-driver-
samples/filesys/fastfat lcov --html fastfat2

Overall coverage rate:
  lines.....: 1.2% (149 of 12573 lines)
```

Initial Run Forkprofile

```
[S2E:s2e] user90@issisp:~$ s2e forkprofile fastfat2

# The fork profile shows all the program counters where execution forked:
# process_pid module_path:address fork_count source_file:line_number (function_name)
00000 /Windows/system32/DRIVERS/imdisk.sys:0x00014a81   15 (no debug info)
00000 /Windows/system32/DRIVERS/imdisk.sys:0x00014a77   14 (no debug info)
00000 /Windows/system32/DRIVERS/imdisk.sys:0x00014a4c    1 (no debug info)
00000 /Windows/system32/DRIVERS/imdisk.sys:0x00014a6c    1 (no debug info)
00000 /Windows/system32/DRIVERS/imdisk.sys:0x00014a58    1 (no debug info)
00000 /Windows/system32/DRIVERS/imdisk.sys:0x00014a3e    1 (no debug info)
```

- Execution seems stuck in the ramdisk driver
  The FAT driver does not get to see the image

- Symbolic input file is by default 256 bytes
  Too small, will never reach the driver?

Increasing Input Size

```
$ nano ~/s2e/projects/fastfat2/bootstrap.sh

function prepare_inputs {
   …
   /c/Python27/python.exe -c \
     "fp = open(\"${SYMB_FILE}\", 'wb');
      fp.write('x'*256)"
}

$ ./launch-s2e.sh

… wait for a few states to fork…

$ killall -9 qemu-system-i386
```

Coverage & Forkprofile

```
$ s2e coverage --sympath ~/s2e/source/windows-driver-samples/filesys/
fastfat lcov --html fastfat2

Overall coverage rate:
  lines.....: 6.4% (800 of 12573 lines)
(Will get to 16% after 3 hours)
```

```
[S2E:s2e] user90@ississp:~/s2e/projects/fastfat2$ s2e forkprofile fastfat2
INFO: [symbols] Looking for debug information for /home/user90/s2e/projects/fastfat2/guestfs/windows/system32/drivers/fs_rec.sys
INFO: [symbols] Looking for debug information for /home/user90/s2e/projects/fastfat2/guestfs/windows/system32/drivers/imdisk.sys
INFO: [symbols] Looking for debug information for /home/user90/s2e/projects/fastfat2/fastfat2.sys
INFO: [forkprofile] # The fork profile shows all the program counters where execution forked:
INFO: [forkprofile] # process_pid module_path:address fork_count source_file:line_number (function_name)
INFO: [forkprofile] 00000 /Windows/System32/Drivers/Fs_Rec.SYS:0x00010a0c   55 (no debug info)
INFO: [forkprofile] 00000 :0x80a700bb   16 (no debug info)
INFO: [forkprofile] 00000 /Windows/System32/Drivers/Fs_Rec.SYS:0x00010ab5    9 (no debug info)
INFO: [forkprofile] 00000 :0x80a700c5    4 (no debug info)
INFO: [forkprofile] 00000 /Windows/system32/DRIVERS/fastfat2.sys:0x00403ab8    4 c:\users\s2e\user90\fsctrl.c:2537 (None)
INFO: [forkprofile] 00000 /Windows/system32/DRIVERS/fastfat2.sys:0x00403aa2    4 c:\users\s2e\user90\fsctrl.c:2537 (None)
INFO: [forkprofile] 00000 /Windows/System32/Drivers/Fs_Rec.SYS:0x00010b1a    3 (no debug info)
INFO: [forkprofile] 00000 /Windows/System32/Drivers/Fs_Rec.SYS:0x00010b16    3 (no debug info)
INFO: [forkprofile] 00000 /Windows/system32/DRIVERS/imdisk.sys:0x00014a81    3 (no debug info)
INFO: [forkprofile] 00000 /Windows/system32/DRIVERS/imdisk.sys:0x00015edb    3 (no debug info)
INFO: [forkprofile] 00000 /Windows/system32/DRIVERS/fastfat2.sys:0x00403acc    3 c:\users\s2e\user90\fsctrl.c:2537 (None)
INFO: [forkprofile] 00000 /Windows/system32/DRIVERS/imdisk.sys:0x00014a77    2 (no debug info)
INFO: [forkprofile] 00000 /Windows/system32/DRIVERS/imdisk.sys:0x00015a75    2 (no debug info)
INFO: [forkprofile] 00000 /Windows/system32/DRIVERS/fastfat2.sys:0x00403a84    2 c:\users\s2e\user90\fsctrl.c:2534 (None)
INFO: [forkprofile] 00000 /Windows/System32/Drivers/Fs_Rec.SYS:0x00010b06    1 (no debug info)
INFO: [forkprofile] 00000 /Windows/system32/DRIVERS/imdisk.sys:0x00014a4c    1 (no debug info)
INFO: [forkprofile] 00000 /Windows/system32/DRIVERS/imdisk.sys:0x00014a6c    1 (no debug info)
INFO: [forkprofile] 00000 /Windows/system32/DRIVERS/imdisk.sys:0x00015a57    1 (no debug info)
INFO: [forkprofile] 00000 /Windows/system32/DRIVERS/imdisk.sys:0x00014a58    1 (no debug info)
```

Execution Statistics

`~/s2e/projects/fastfat2/s2e-last/run.stats`

- Low path throughput
  After 10 minutes, ~14 paths completed, ~95 left
- Paths in S2E may run for a long time
  Overhead of running an entire system in DBT
  In the future: use hardware virtualization

- Tip: use the smallest system possible for your app
  No point in running a simple binary on Windows 10 if it can run just as well on WinXP (or Win10 IoT core).

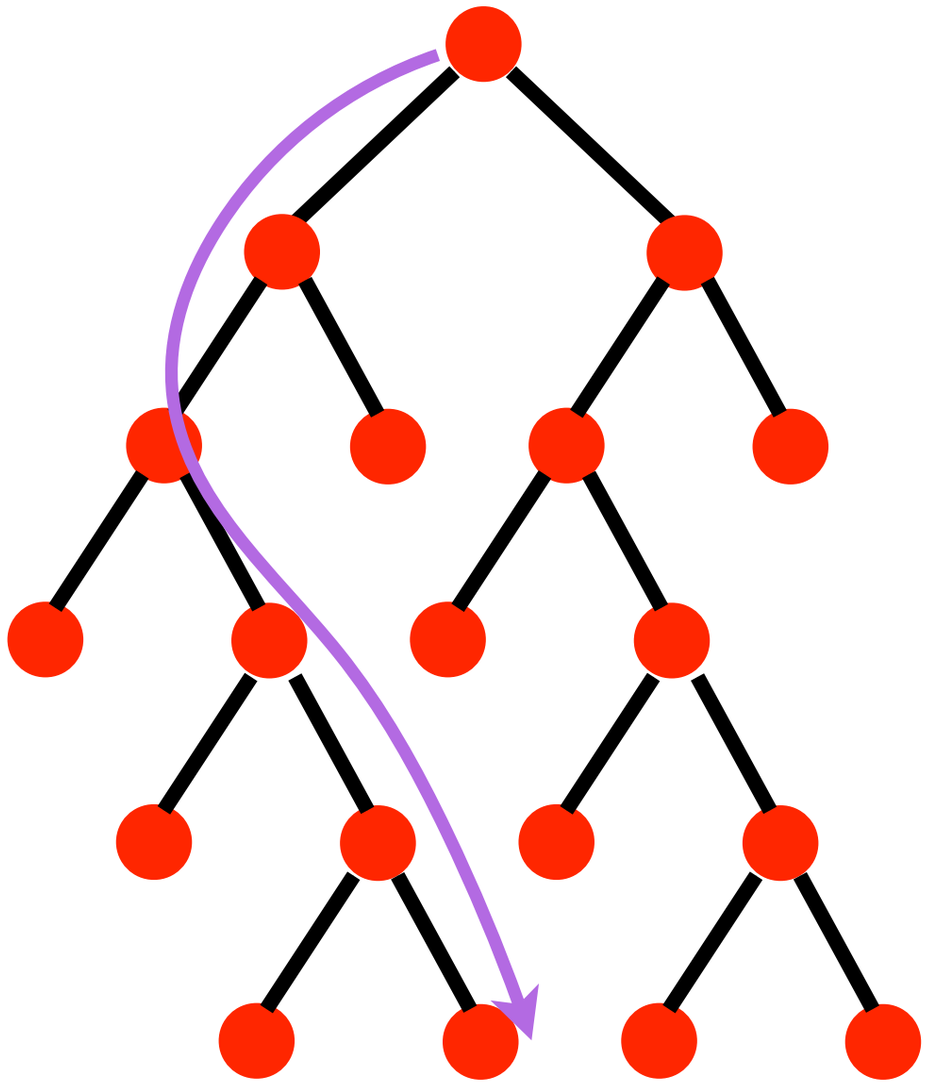
Parallelizing Execution

- S2E can split the workload across multiple cores
- If an S2E instance has 10 states, it will fork off a new instance that will take half of the states
- Load balancing happens every few seconds if there are free cores available
- Increase the value of S2E_MAX_PROCESSES variable in launch-s2e.sh

Coverage & Forkprofile

- After 10 minutes with max 10 cores
- ~170 states forked, 45 completed
- Forked in fastfat2.sys, fs_rec.sys, imdisk.sys and other kernel components
- Coverage went to 6.4% (and will go to 16% after 3 hours)

Concolic Execution



Use a well-formed image file as a source of concrete values

Get skeleton path, then fuzz around

Concolic Execution

Works like symbolic execution, but also uses concrete input as hint.

```
int is_image_valid(char *image)
{
  if (image[0xff] != 0xaa) {
    return -1;
  }

  if (image[0xfe] != 0x55) {
    return -1;
  }

  return 0;
}
```

(λ_{511}; 0xaa)

(λ_{510}; 0x55)

First explore the path where the last two bytes are 0x55 0xaa

Seed Files

- Generate a few valid disk images
  Various sizes of FAT, directories, files, long file names, etc.
- Place them into the `seeds` folder of the S2E project
- S2E will use these seeds to find more paths and generate more test cases
- Seeds could come from a fuzzer and test cases generated by S2E could go back to the fuzzer

Seed Files

```
# bootstrap.sh
while true; do
  if get_seed_file; then
     break
  fi
done

     execute_target $target $seed_file
```

- State 0 is the seed fetching state. It does not run the binary.
- State 0 forks a new state when there is a new seed. This new state will run the binary on the seed file.
- The SeedSearcher plugin switches to state 0 when there is a new seed available

Using Seed Files with S2E

```
$ s2e new_project -n fastfat-seeds --use-seeds ~/
s2e/source/windows-driver-samples/filesys/fastfat/
Debug/fastfat2.inf @@
```

```
$ wget -O make-seeds.sh https://pastebin.com/raw/
Y75JL0VT

$ dos2unix make-seeds.sh && chmod +x make-seeds.sh

$ ./make-seeds.sh s2e/projects/fastfat-seeds/seeds
```

Using Seed Files with S2E

```
[S2E:s2e] user90@issisp:~/s2e/projects/fastfat-seeds$ ls -1
seeds/
0-0.fat12.img
0-0.fat12.img.symranges
1-0.fat16.img
1-0.fat16.img.symranges
2-0.fat32.img
2-0.fat32.img.symranges
```

***.symranges files specify which byte ranges will be made symbolic.**
If the file is empty, the seed file will be entirely concrete.

Using Seed Files with S2E

```
$ nano ~/s2e/projects/fastfat2/bootstrap.sh
```

```
function execute_target {
    local TARGET
    local SYMB_FILE

    TARGET="$1"
    SYMB_FILE="$2"

    # Activate fault injection right before loading the driver
    ./drvctl.exe set_config FaultInjectionActive 0

    # Ask windows to load the driver
    install_driver "$(win_path "$TARGET")"

    sc start fastfat2

    imdisk -a -o hd -f x:\\input -m F:
    mount -a
}
```

Using Seed Files with S2E

```
$ s2e coverage --sympath ~/s2e/source/windows-driver-
samples/filesys/fastfat lcov --html fastfat-seeds
```

```
Overall coverage rate:
  lines......: 15.1% (1894 of 12573 lines)
```

Using Seed Files with S2E

```
$ nano ~/s2e/projects/fastfat2/bootstrap.sh
function execute_target {
    …

    sc start fastfat2

    imdisk -a -o hd -f x:\\input -m F:
    mount -a

    ls -l /f
    echo bla > /f/test.txt

    run_cmd "chkdsk f:"

    find /f -exec cat {} \; > /dev/null

    rm -rf /f/*
    echo asldfj > "/f/this is a very long file name.txt"

    imdisk -D -m F:
    sc stop fastfat2
}
```

Using Seed Files with S2E

```
$ s2e coverage --sympath ~/s2e/source/windows-driver-
samples/filesys/fastfat lcov --html fastfat-seeds

Overall coverage rate:
  lines......: 40.5% (5098 of 12573 lines)
```

Symbolic Files

- Let's make some bytes in the image file symbolic
- How about making file attributes and starting cluster symbolic?
- Requires some knowledge of the format of directory entries
- https://en.wikipedia.org/wiki/Design_of_the_FAT_file_system

Symbolic Files

```
$ hexdump -C seeds/0-0.fat12.img | less

00002600  41 66 00 61 00 73 00 74  00 66 00 0f 00 85 61 00  |Af.a.s.t.f....a.|
00002610  74 00 2d 00 73 00 65 00  65 00 00 00 64 00 73 00  |t.-.s.e.e...d.s.|
00002620  46 41 53 54 46 41 7e 31  20 20 20 10 00 00 4d 90  |FASTFA~1   ...M.|
00002630  e4 4c e4 4c 00 00 4d 90  e4 4c 02 00 00 00 00 00  |.L.L..M..L......|
00002640  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

seeds/0-0.fat12.img.symranges

```
# Make file attributes symbolic
0x260b-1

# Starting cluster
0x261a-2
```

Results

```
$ s2e coverage --sympath ~/s2e/source/windows-driver-
samples/filesys/fastfat lcov --html fastfat-seeds

Overall coverage rate:
  lines......: 41.5% (5219 of 12573 lines)
```

```
# The fork profile shows all the program counters where execution forked:
# process_pid module_path:address fork_count source_file:line_number (function_name)
00000 :0x807e1272   56 (no debug info)
01400 /WINDOWS/system32/ufat.dll:0x5b2538b9   34 (no debug info)
01400 /WINDOWS/system32/ufat.dll:0x5b2538ce   28 (no debug info)
01400 /WINDOWS/system32/ufat.dll:0x5b259c3d   20 (no debug info)
01400 /WINDOWS/system32/ufat.dll:0x5b257188    8 (no debug info)
01400 /WINDOWS/system32/ufat.dll:0x5b2538b3    8 (no debug info)
01400 /WINDOWS/system32/ufat.dll:0x5b251a76    8 (no debug info)
01400 /WINDOWS/system32/ufat.dll:0x5b251a7b    8 (no debug info)
01260 /msys/1.0/bin/msys-1.0.dll:0x7100605b    2 (no debug info)
01760 /s2e/s2ecmd.exe:0x00401f59    1 /home/issisp/s2e/source/s2e/guest/common/s2ecmd/s2ecmd.cpp:237
(handler_get_seed_file)
00000 /Windows/system32/DRIVERS/fastfat2.sys:0x004231e2    1 c:\users\s2e\user90\dirsup.c:1468 (None)
00000 /Windows/system32/DRIVERS/fastfat2.sys:0x00423036    1 c:\users\s2e\user90\dirsup.c:1363 (None)
00000 /Windows/system32/DRIVERS/fastfat2.sys:0x00423939    1 c:\users\s2e\user90\dirsup.c:1902 (None)
00000 /Windows/system32/DRIVERS/fastfat2.sys:0x0041f1bb    1 c:\users\s2e\user90\dirctrl.c:930 (None)
…
```

Limiting Forks to Select Modules

```
-- This configuration shows an example that kills states if they fork in
-- a specific module.
pluginsConfig.LuaCoreEvents = {
    -- This annotation is called in case of a fork. It should return true
    -- to allow the fork and false to prevent it.
    onStateForkDecide = "onStateForkDecide"
}

function onStateForkDecide(state)
   mmap = g_s2e:getPlugin("ModuleMap")
   mod = mmap:getModule(state)
   if mod == nil then
     g_s2e:info("module is null, preventing forking")
     return false
   end

   name = mod:getName(state)
   if name == "s2ecmd.exe" or name == “s2e.sys" then
     g_s2e:info("module is s2ecmd or s2e.sys, allowing forking")
     return true
   end

   if name == "fastfat2.sys" then
     return true
   end

   return false
end
```

Testing Error Recovery Code

```
    RtlInitUnicodeString( &UnicodeString, L"\\Fat2" );
    Status = IoCreateDevice( DriverObject,
                             0,
                             &UnicodeString,
                             FILE_DEVICE_DISK_FILE_SYSTEM,
                             0,
                             FALSE,
                             &FatDiskFileSystemDeviceObject );

    if (!NT_SUCCESS( Status )) {
        return Status;
    }
```

```
 Status = FsRtlRegisterFileSystemFilterCallbacks( DriverObject,
                                                  &FilterCallbacks );

 if (!NT_SUCCESS( Status )) {

     IoDeleteDevice( FatDiskFileSystemDeviceObject );
     IoDeleteDevice( FatCdromFileSystemDeviceObject );
     return Status;
 }
```

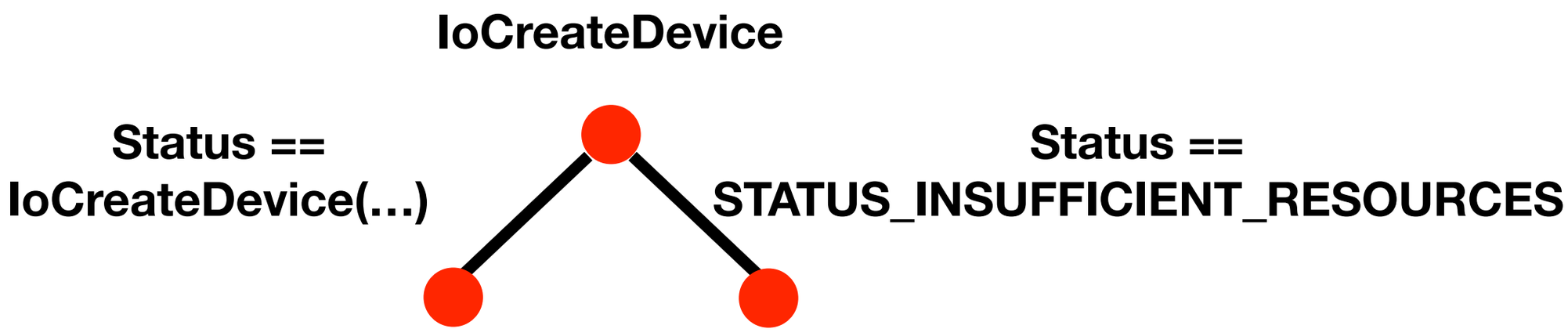
Fault Injection

- Windows has built-in tools to simulate various fault scenarios for drivers
- Injects faults probabilistically, does not support all APIs
- We can improve this with symbolic fault injection

Symbolic Fault Injection

```
RtlInitUnicodeString( &UnicodeString, L"\\Fat2" );
Status = IoCreateDevice( DriverObject,
                         0,
                         &UnicodeString,
                         FILE_DEVICE_DISK_FILE_SYSTEM,
                         0,
                         FALSE,
                         &FatDiskFileSystemDeviceObject );

if (!NT_SUCCESS( Status )) {
    return Status;
}
```



API Annotations

- Intercept API calls of the module under test `GuestCodeHooking` plugin + `s2e.sys` driver

- Annotations can either run in the guest or in an plugin
  Much easier to write them in the guest

- Fork a new execution path, first path calls the original API, second path skips the call and directly return an error.

- ~5 lines of code per annotation

Scalability

- No need for constraint solver

- Faults are injected once per call stack
  Finite and predictable number of paths

```
$ nano ~/s2e/projects/fastfat2/bootstrap.sh
```

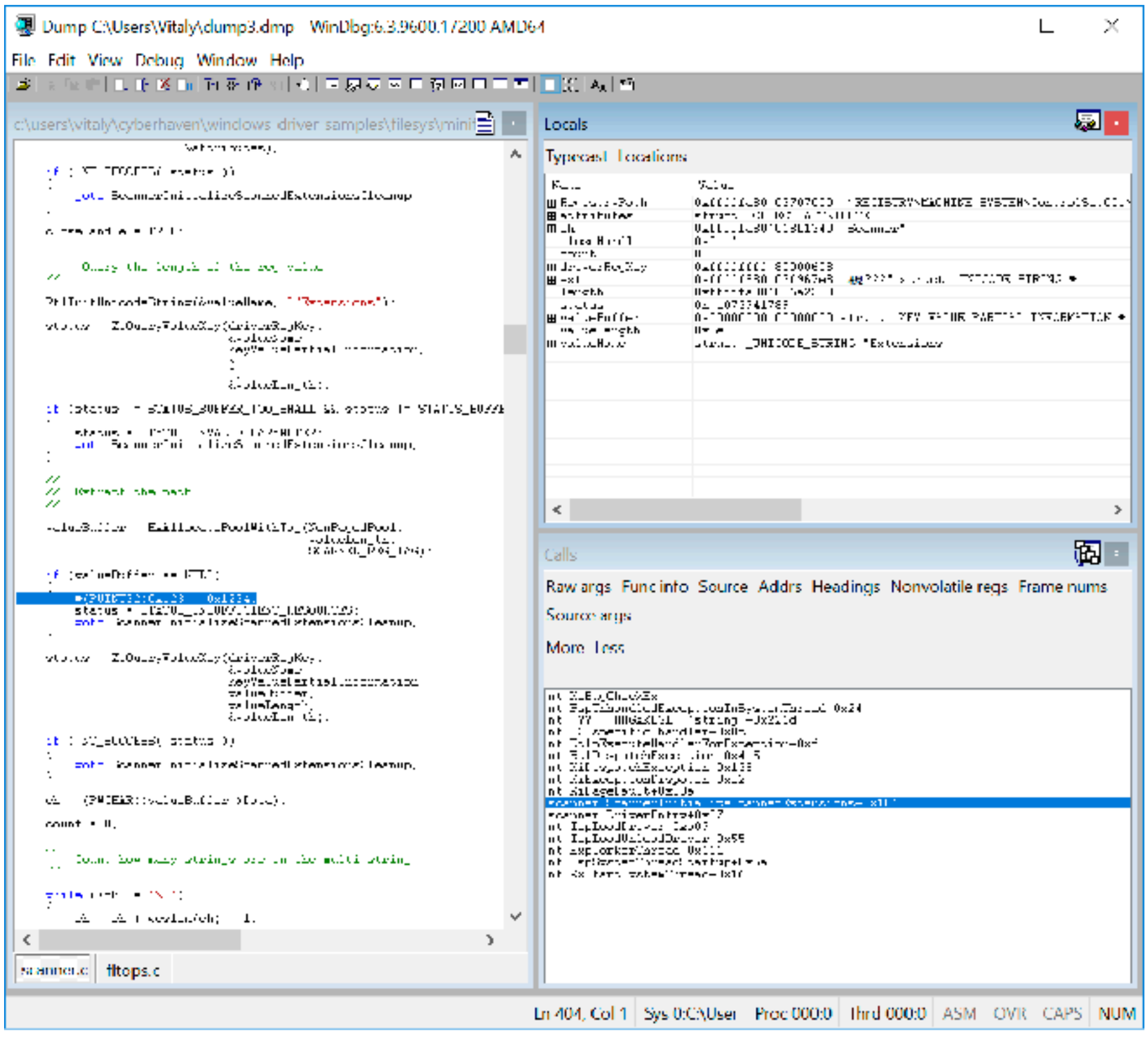
```
function execute_target {
    local TARGET
    local SYMB_FILE

    TARGET="$1"
    SYMB_FILE="$2"

    # Activate fault injection right before loading the driver
    ./drvctl.exe set_config FaultInjectionActive 1

    …
}
```

Crash Dumps


Dump C:\Users\Vitaly\dump3.dmp WinDbg:6.3.9600.17200 AMD64
File Edit View Debug Window Help
c:\users\vitaly\cyberhaven\windows driver samples\filesys\mini
Locals
Typecast Locations
Calls
Raw args Func info Source Addrs Headings Nonvolatile regs Frame nums
Source args
More Less
scanner.c
fltops.c
Ln 404, Col 1
Sys 0:C:\User
Proc 000:0
Thrd 000:0
ASM
OVR
CAPS
NUM

Crash Dumps

```
# s2e-config.lua

add_plugin(“WindowsCrashMonitor")
pluginsConfig.WindowsCrashMonitor = {
    terminateOnCrash = true,

    -- Make this true if you want crashes.
    -- Note that crashes may be very large (100s of MBs)
    generateCrashDump = true,

    -- Limit number of crashes we generate
    maxCrashDumps = 1,

    -- Uncompressed dumps have the same size as guest memory
    -- you almost always want to compress them.
    compressDumps = true
}
```

Conclusion

- Using S2E for Windows device driver testing
- Dealing with path explosion
- Seeds, parallel symbolic execution, fault injection, code coverage, fork profiling

We are hiring!

https://s2e.systems

Ready-for-use docker image, demos, tutorials, source code, documentation