MWR LABS

Platform Agnostic Kernel Fuzzing

James Loureiro and Georgi Geshev

MWR InfoSecurity

MWR LABS

What is this talk about?

- Motivation
- Fuzzer Architecture
- Kernel Fuzzing Caveats
- Case Study: Windows
- Other Operating Systems
- Results
- Further Work

MWR LABS

Motivation

- Kernels are still considered weak spots.
- Privilege escalations become necessary commodity.
- Friendly internal competition.
  - Nils' Windows Kernel Fuzzing presented at T2 in October 2015.
- Improving general OS security.
- Plan evolved into developing a platform agnostic framework.

MWR LABS

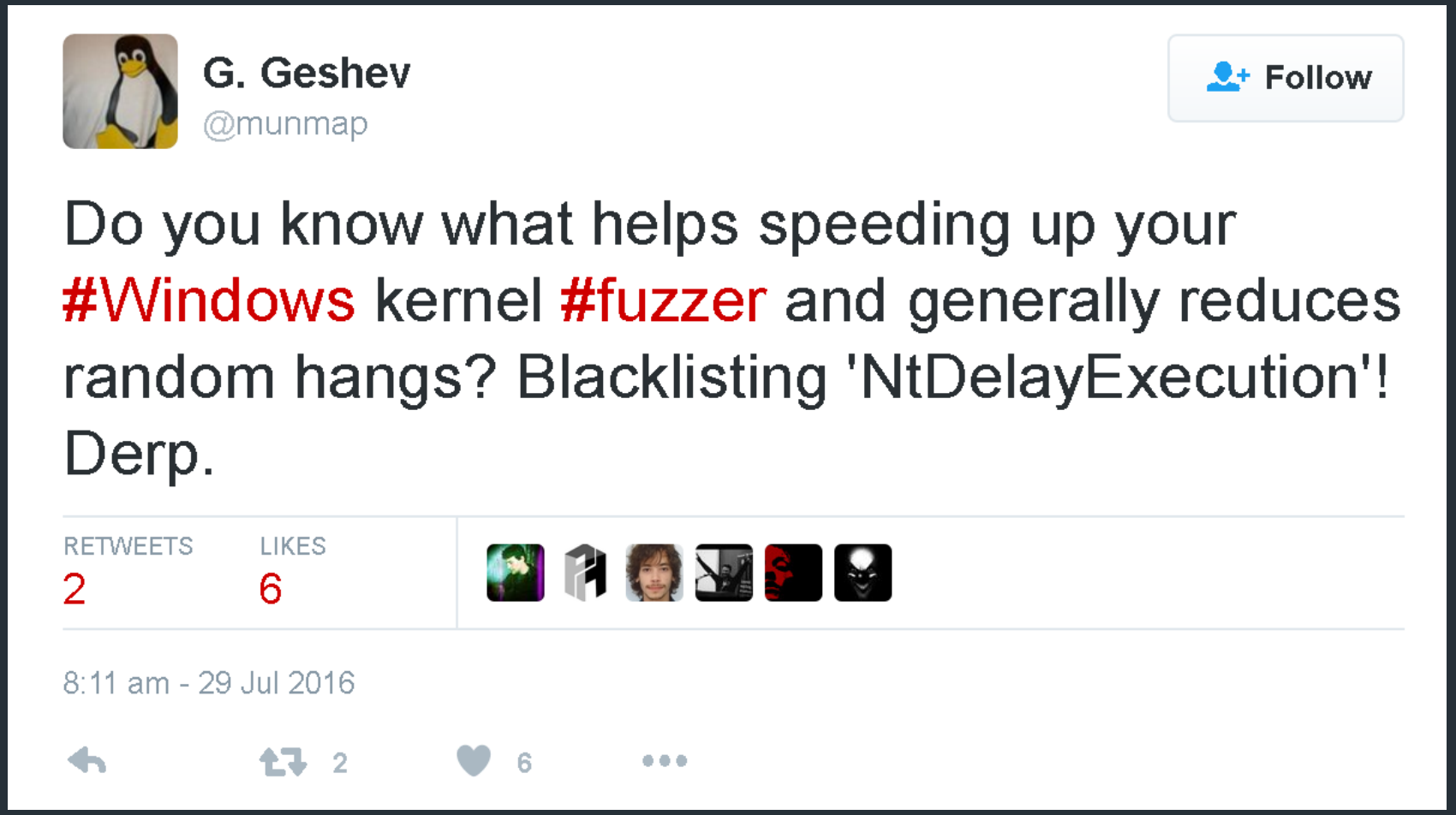
How hard can it be?

MWR LABS

Results: Windows

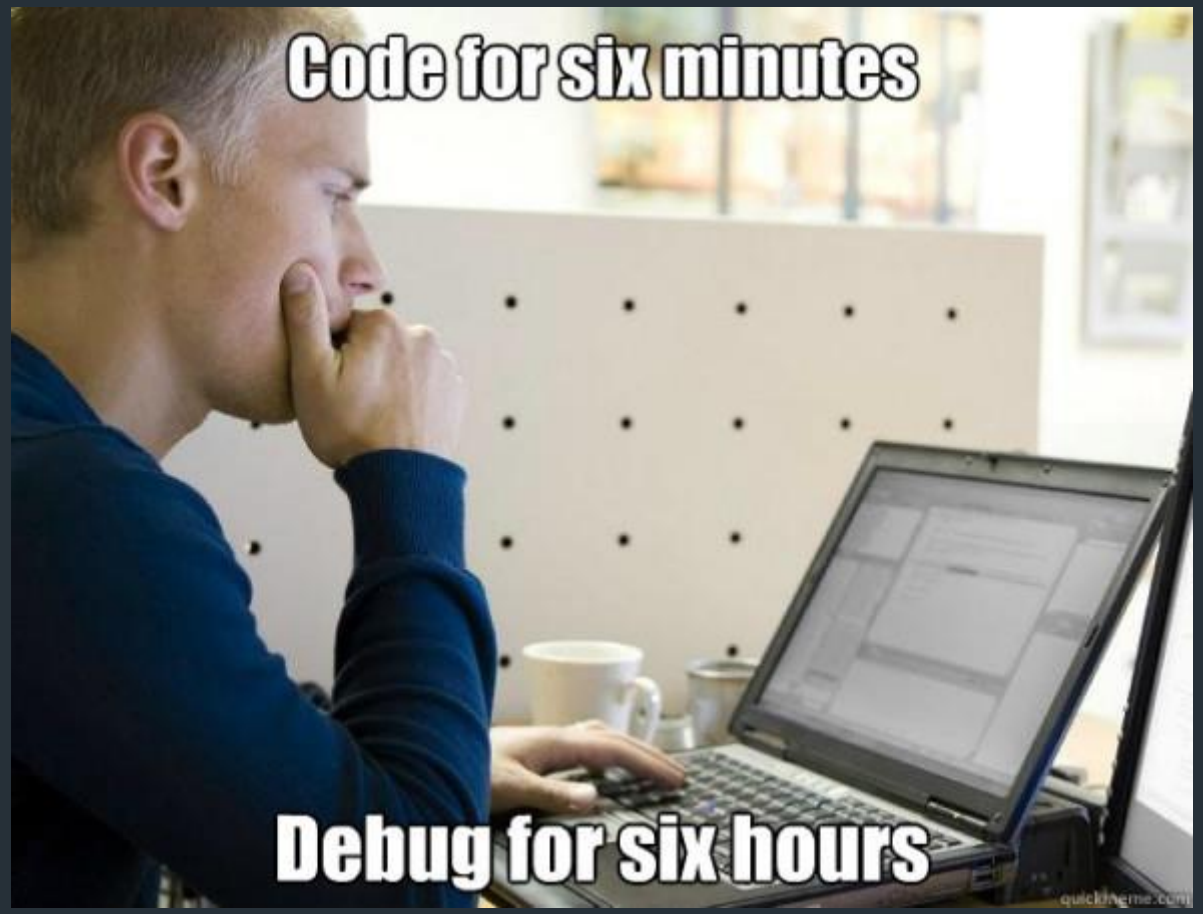
Number of VMs	16
Hours	48
Total Number of Crashes	65
Unique Crashes	13

Results: Windows



MWR LABS

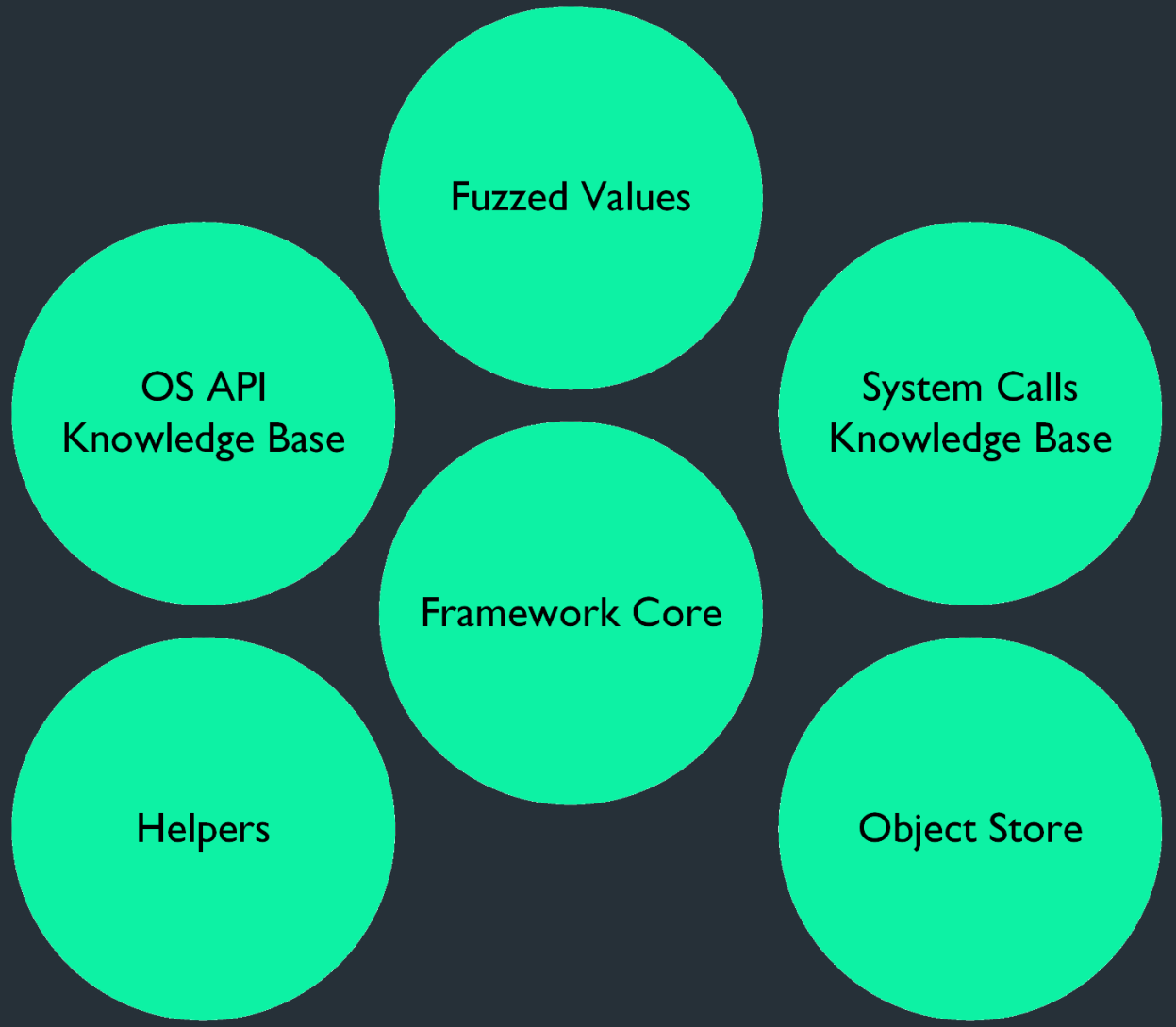
Framework Release



MWR LABS

Fuzzer Architecture

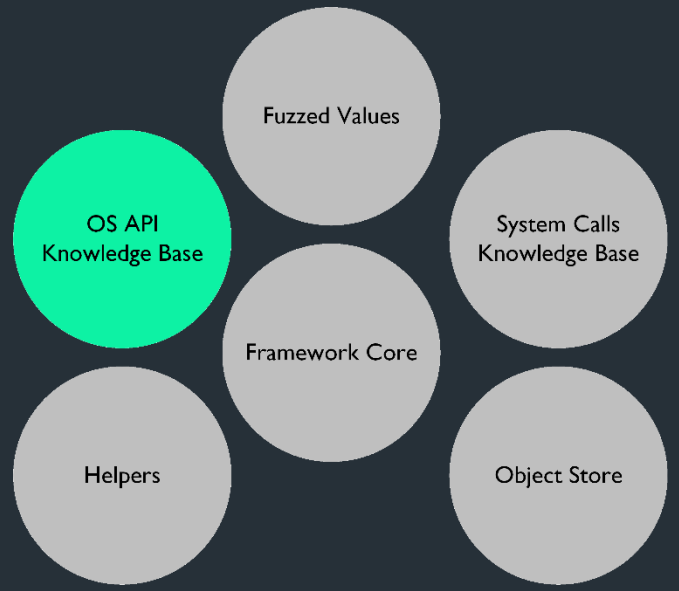
- Nicely decoupled components.
  - Knowledge Bases
  - Object Store
  - Helper Functions
  - Fuzzed Values Generator
- Original prototype in Python.
- Rewritten in C for efficiency.



MWR LABS

OS API Knowledge Base

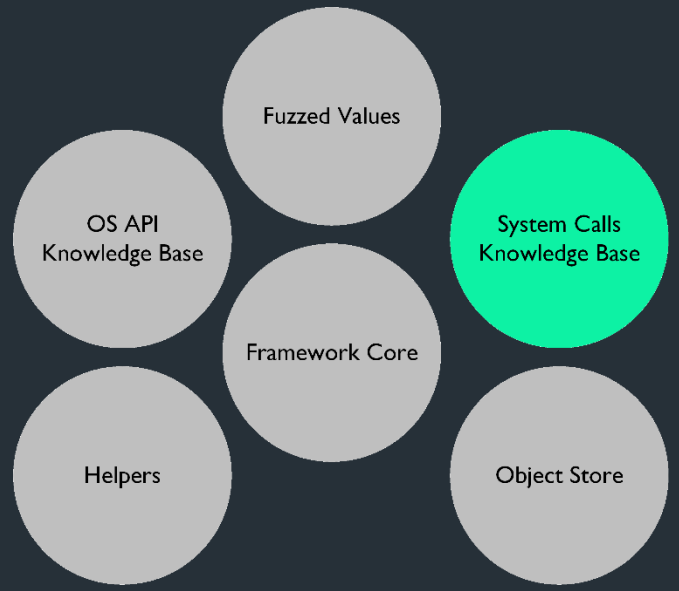
- OS APIs to interact with system libraries.
  - File and Data Access
  - User Interface
  - Graphics and Multimedia
  - Devices
  - Networking
- Many of these wrap system calls.
- List of OS-specific API calls can be ‘plugged’ into the framework.



MWR LABS

System Calls Knowledge Base

- User space to kernel space communication.
  - Requesting resources and actions.
- Implemented and handled in kernel-land.
- Low-level OS-specific interaction.
  - Requires OS- and architecture-specific assembly.
- List of OS-specific system calls can be 'plugged' into the framework.



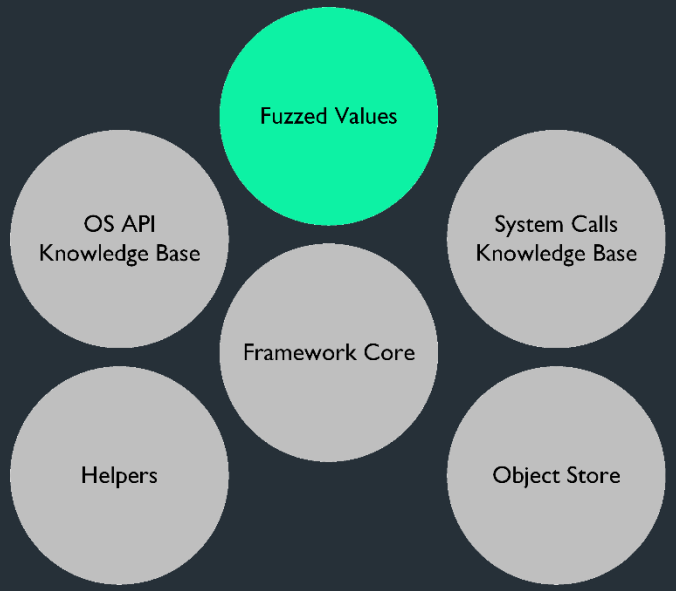

```
typedef struct
{
    unsigned int scid;
    DATATYPE argument_datatypes[32];
    DATATYPE return_datatype;
} SYSCALL;
```

MWR LABS

Fuzzed Values Generator

- Functions return fuzzed basic data types.
  - Boolean, integer, floating point, etc.

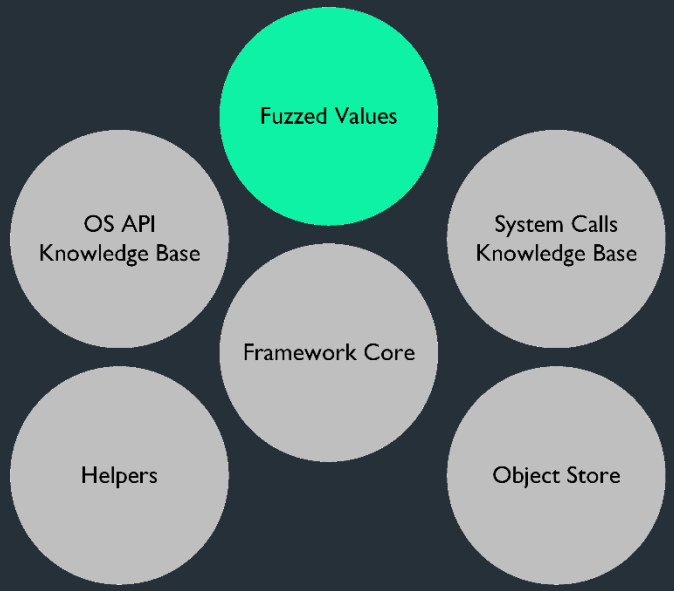
```
bool_t   get_fuzzed_bool (void);
char8_t  get_fuzzed_char8 (void);
char16_t get_fuzzed_char16 (void);
int8_t   get_fuzzed_int8 (void);
int16_t  get_fuzzed_int16 (void);
int32_t  get_fuzzed_int32 (void);
int64_t  get_fuzzed_int64 (void);
uint8_t  get_fuzzed_uint8 (void);
uint16_t get_fuzzed_uint16 (void);
uint32_t get_fuzzed_uint32 (void);
uint64_t get_fuzzed_uint64 (void);
real32_t get_fuzzed_real32 (void);
real64_t get_fuzzed_real64 (void);
```



MWR LABS

Fuzzed Values Generator

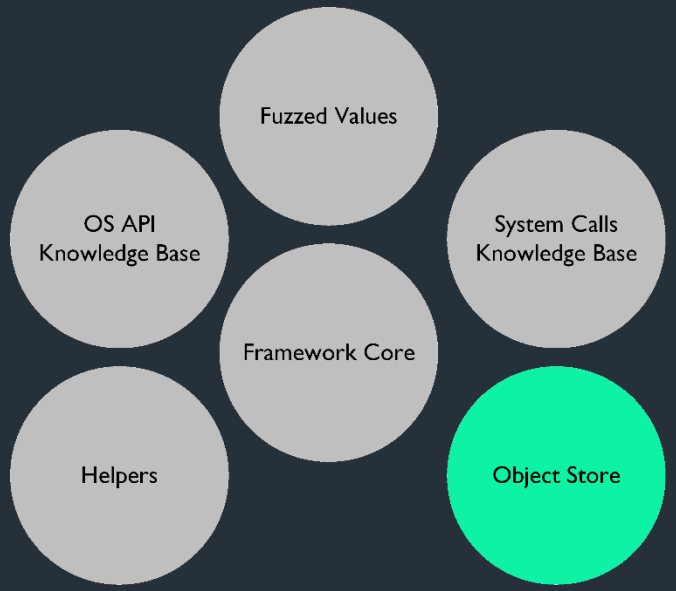
- Random but not ‘too random’.
- Calls fail when arguments don’t make sense.
- Predefined list of ‘good’ values per data type.
- Increases likelihood of succeeding.



MWR LABS

Object Store

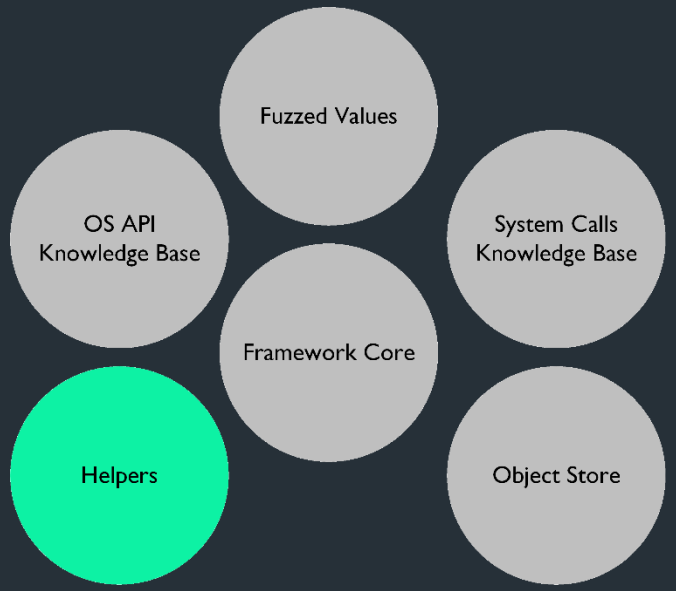
- Maintaining state across the calls.
- Preserving OS-specific objects of interest.
- Deterministically populated by the fuzzer.
- Retrieving, updating, and inserting objects.
- Implemented as a global array of objects.



MWR LABS

Helper Functions

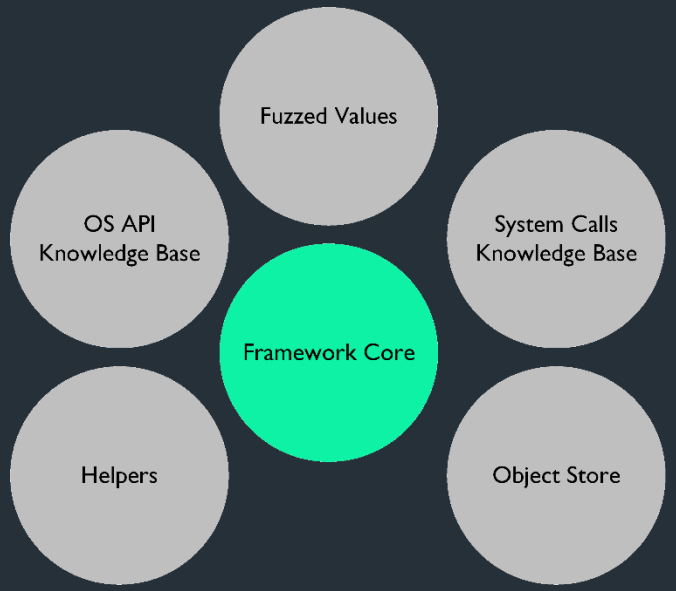
- Library calls expect valid OS-specific structures.
- Generate, populate and return valid structures.
- Populated with ‘mostly’ valid data.



MWR LABS

Logging

- PRNG Seed
  - Log the PRNG seed on startup.
  - Does not generate a standalone test case.
  - Replay the fuzzer run with same seed.
- Logging C statements, i.e. logs are source files.
  - Log files are fed into a template.
  - Compiles an identical test case.
- Potential Problems
  - Flushing to Disk

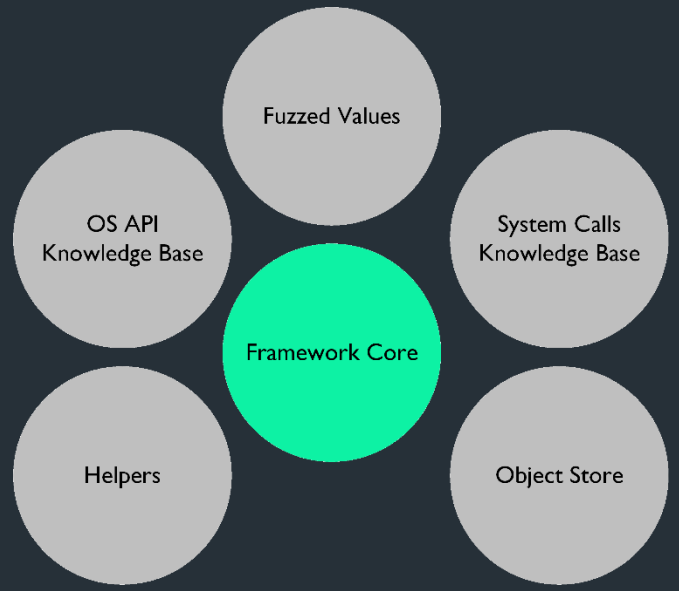


MWR LABS

Logging (cont.)

- Tedious!

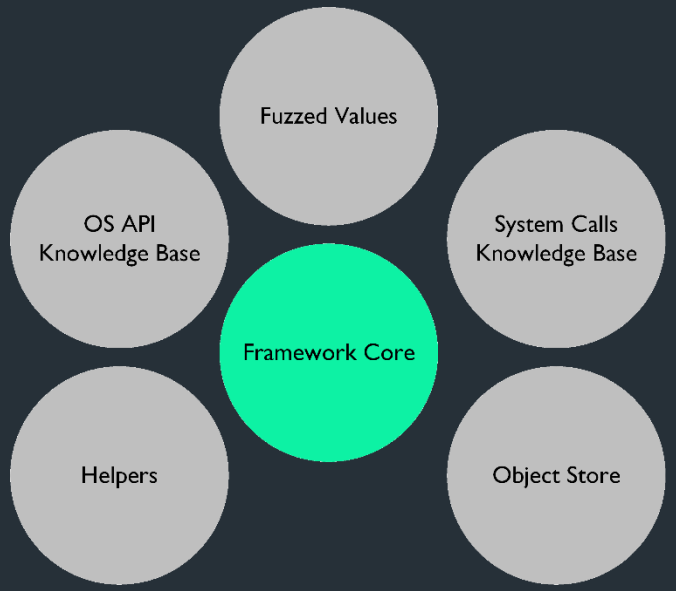
```
// Declare the variable(s).
int x_fn_name;
// Generate a globally unique variable ID.
char vid[16];
sprintf(vid, "%d%d", get_time_in_ms(), rand() % 1024);
// Log the declaration.
logger("int x_fn_name%s;", vid);
// Assign value and log.
x_fn_name = get_fuzzed_int32();
logger("x_fn_name%s = %d;", vid, x_fn_name);
// Make a call.
logger("fn_call(x_fn_name%s);", vid);
fn_call(x_fn_name);
```



MWR LABS

Crash Detection

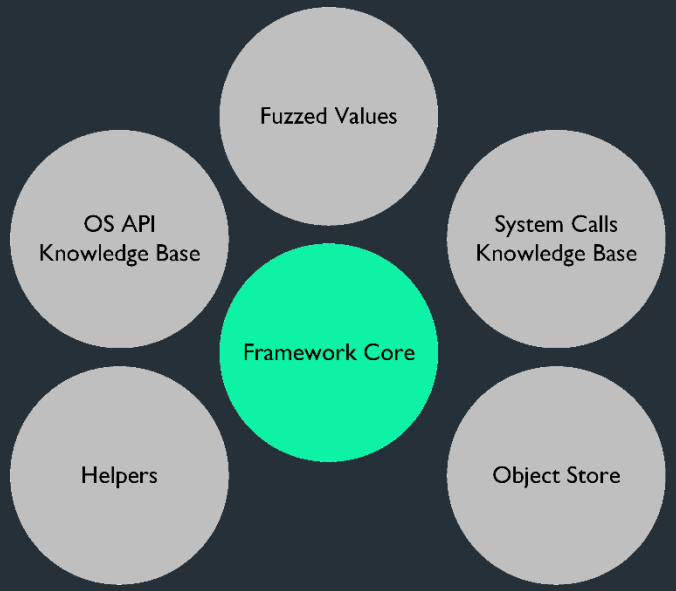
- Attaching a kernel debugger while executing.
  - Requires one or more debugger processes.
  - Slower execution.
  - Crashes are instantly detected and analysed.
- Unattended execution.
  - Letting the OS handle the crash on its own.
  - Faster execution.
  - Crash details are recovered and analysed upon rebooting.
  - Hypervisor logs can be used for detecting a VM crash.



MWR LABS

Crash Detection (cont.)

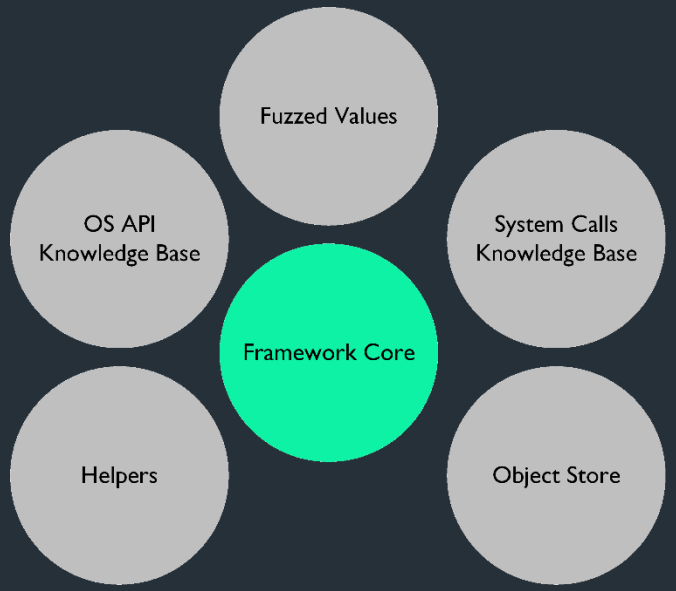
- Search for a memory dump upon rebooting.
- Log file will be matched with a memory dump.
- Memory dumps, if any, are analysed locally.
- Fuzzer log, memory dump, and memory dump analysis are timestamped and archived.



MWR LABS

Crash Storing and Triaging

- Crash details are fed into a central database.
  - Fuzzer Log
  - OS Crash Information
  - Memory Dump (Windows)
- Deduplication
  - Based on stack traces.
- Categorisation
  - Based on several crash characteristics.
  - Type of AV, Faulting IP, Bug Check ID, etc.



MWR LABS

From Zero to Zero-Day

- Bootstrap the worker(s) and reboot.
  - Python script for installing prerequisites and fine-tuning the system.
- Looks for OS crash logs.
  - Collect memory dump(s), if any, and analyse.
  - Submit logs, and memory dump analysis to a central DB.
- Fuzzer kicks in on logon/startup.
- Populate the object store.

MWR LABS

From Zero to Zero-Day (cont.)

- For a predefined number of iterations, pick up a library or system call.
  - Prepare arguments using fuzzed values, structs from helper functions or objects from the object store.
  - Invoke the library/system call, check for success and optionally insert returned object to object store if it is valid.
- Clean up any temporary files, remove log, and revert to a clean state.
  - Rebooting should guarantee a clean state.
  - Reverting back to a clean VM snapshot is better.

MWR LABS

Kernel Fuzzing Caveats

- Crashing Hypervisors
  - Fuzzing in the Cloud
  - Nesting Hypervisors
- Protecting Fuzzer from Itself
- VM Monitoring and Management

MWR LABS

Case Study: Fuzzing Windows

- Original Effort and Main Focus of the Project
- OS-specific Tweaks
- Knowledge Base Implementation Details
- Several Examples
  - Object Store
  - System Calls
  - OS API Calls
  - Helper Functions
- Bootstrapping a Windows Fuzzing VM

Case Study: Fuzzing Windows Attack Surface

MWR LABS

- WIN32K.SYS Kernel-Mode Driver
  - Window Manager
    - Desktops, Windows, Menus, Cursors, etc.
  - Graphic Device Interface (GDI)
    - Bitmaps, Brushes, Colors, Fonts, Pens, etc.
  - DirectX Thunks
- WIN32K User-Mode Libraries
  - USER32.DLL, IMM32.DLL
  - GDI32.DLL, MSIMG32.DLL

Case Study: Fuzzing Windows Attack Surface

MWR LABS

- Objects
  - Data Structures Representing System Resources
  - Files, processes, events, windows, fonts, etc.
- Object Categories
  - User
  - GDI
  - Kernel
- Object Handles

Case Study: Fuzzing Windows Object Store

MWR LABS

- Object Handles

WinNT.h

```
typedef void *HANDLE;
// …
#define DECLARE_HANDLE(name) typedef HANDLE name
```

WinDef.h

```
DECLARE_HANDLE              (HWND);
DECLARE_HANDLE              (HHOOK);
// …
DECLARE_HANDLE              (HGDIOBJ);
DECLARE_HANDLE              (HKEY);
DECLARE_HANDLE              (HBITMAP);
DECLARE_HANDLE              (HBRUSH);
DECLARE_HANDLE              (HDC);
```

Case Study: Fuzzing Windows Object Store

MWR LABS

- Fuzzer Object Store
  - We keep track of handles to various objects.
  - Handles are retrieved and consumed by both library and system calls.

handles_database.h

```
HANDLE HANDLES[128];
char* HANDLE_CREATOR[128];
```

bughunt.h

```
typedef struct {
    HANDLE value;
    int index;
} BH_Handle;
```

Case Study: Fuzzing Windows Object Store

MWR LABS

- Fuzzer Object Store

handles_database.h

```
// Populate our handles database.
void make_HANDLES (void);

// Retrieve a random handle and wrap it in a BH_Handle structure.
BH_Handle get_random_HANDLE (void);

// Retrieve a handle by its index.
HANDLE get_specific_HANDLE (int n);

// Insert a handle to the store.
HANDLE put_random_HANDLE (HANDLE handle, char* HandleCreator);
```

Case Study: Fuzzing Windows System Calls

- System Calls Knowledge Base
  - Reverse Engineering
  - ReactOS
- Assembly Snippets
  - Inline x86 Assembly
  - Standalone x64 Assembly (ML64)

Case Study: Fuzzing Windows System Calls

MWR LABS

- System Calls Knowledge Base
  - Array of System Calls IDs and Corresponding Argument Types

```
SYSCALL SYSCALLS[] =
{
   // …
   { 0x1103, { _HANDLE, _UINT32, _UINT32, NIL }, _NIL },
   { 0x130B, { _HANDLE, _HANDLE, _BOOL, NIL }, _NIL },
   { 0x107F, { _HANDLE, _VOID_PTR, NIL }, _NIL },
   { 0x1119, { _HANDLE, _HANDLE, NIL }, _NIL },
   { 0x1106, { _HANDLE, NIL }, _BOOL },
   // …
}
```

Case Study: Fuzzing Windows System Calls

MWR LABS

- System Calls Invocation Template
  - Takes a System Call ID and System Call Arguments

```
#ifdef _M_IX86

__declspec(noinline) DWORD __stdcall bughunt_syscall ( DWORD SCID, … ) {
   __asm { … }
}

#elif  _M_IX64

extern DWORD __stdcall bughunt_syscall( DWORD SCID, … );

#endif
```

Case Study: Fuzzing Windows System Calls

MWR LABS

- x86
  - Arguments on the Stack
  - EAX Set to System Call ID
  - ‘KiFastSystemCall’
- x64
  - First Four Arguments in Registers
    - RCX, RDX, R8, R9
  - Additional Arguments on Stack
  - RAX Set to System Call ID
  - ‘syscall’

Case Study: Fuzzing Windows OS API Calls

MWR LABS

- OS API Calls Knowledge Base
  - MSDN
- Array of Function Pointers to ~500 Library Call Wrappers

library_calls.h

```
void (*LIBRARY_CALLS[]) () = {
    BH_CreateDialog,
    BH_CreateDialogParam,
    BH_DialogBox,
    // …
    BH_CloseWindow,
    BH_CreateWindow
}
```

Case Study: Fuzzing Windows OS API Calls

MWR LABS

MSDN

```
BOOL WINAPI DestroyCaret(void);
```

caret.h

```
VOID BH_DestroyCaret() {
      logger("DestroyCaret();");
      DestroyCaret();
}
```

Case Study: Fuzzing Windows OS API Calls

MWR LABS

MSDN

```
BOOL WINAPI DestroyCaret(void);
```

caret.h

```
VOID BH_DestroyCaret() {
      logger("DestroyCaret();");
      DestroyCaret();
}
```

Case Study: Fuzzing Windows OS API Calls

MWR LABS

MSDN

```
BOOL WINAPI DestroyCursor(
  _In_ HCURSOR hCursor
);
```

cursor.h

```
VOID BH_DestroyCursor() {
   BH_Handle hCursor_BH_DestroyCursor;
   char vid[16];
   sprintf(vid, "%d%d", get_time_in_ms(), rand() % 1024);
   logger("HANDLE hCursor_BH_DestroyCursor%s;", vid);
   hCursor_BH_DestroyCursor = get_random_HANDLE();
   logger("hCursor_BH_DestroyCursor%s = get_specific_HANDLE(%d);",
          vid, hCursor_BH_DestroyCursor.index);
   logger("DestroyCursor(hCursor_BH_DestroyCursor%s);", vid);
   DestroyCursor(hCursor_BH_DestroyCursor.value);
}
```

Case Study: Fuzzing Windows OS API Calls

MWR LABS

MSDN

```
BOOL WINAPI DestroyCursor(
  _In_ HCURSOR hCursor
);
```

cursor.h

```
VOID BH_DestroyCursor() {
   BH_Handle hCursor_BH_DestroyCursor;
   char vid[16];
   sprintf(vid, "%d%d", get_time_in_ms(), rand() % 1024);
   logger("HANDLE hCursor_BH_DestroyCursor%s;", vid);
   hCursor_BH_DestroyCursor = get_random_HANDLE();
   logger("hCursor_BH_DestroyCursor%s = get_specific_HANDLE(%d);",
          vid, hCursor_BH_DestroyCursor.index);
   logger("DestroyCursor(hCursor_BH_DestroyCursor%s);", vid);
   DestroyCursor(hCursor_BH_DestroyCursor.value);
}
```

MSDN

```
COLORREF SetDCPenColor(
  _In_ HDC      hdc,
  _In_ COLORREF crColor
);
```

pen.h

```
VOID BH_SetDCPenColor() {
   BH_Handle hdc_BH_SetDCPenColor;
   COLORREF cr_BH_SetDCPenColor;
   char vid[16];
   sprintf(vid, "%d%d", get_time_in_ms(), rand() % 1024);
   logger("HANDLE hdc_BH_SetDCPenColor%s;", vid);
   hdc_BH_SetDCPenColor = get_random_HANDLE();
   logger("hdc_BH_SetDCPenColor%s = get_specific_HANDLE(%d);", vid,
          hdc_BH_SetDCPenColor.index);
   cr_BH_SetDCPenColor = get_COLORREF(vid);
   logger("SetDCPenColor(hdc_BH_SetDCPenColor%s, cr%s);", vid, vid);
   SetDCPenColor(hdc_BH_SetDCPenColor.value, cr_BH_SetDCPenColor);
}
```

MWR LABS

MSDN

```
COLORREF SetDCPenColor(
  _In_ HDC      hdc,
  _In_ COLORREF crColor
);
```

pen.h

```
VOID BH_SetDCPenColor() {
    BH_Handle hdc_BH_SetDCPenColor;
    COLORREF cr_BH_SetDCPenColor;
    char vid[16];
    sprintf(vid, "%d%d", get_time_in_ms(), rand() % 1024);
    logger("HANDLE hdc_BH_SetDCPenColor%s;", vid);
    hdc_BH_SetDCPenColor = get_random_HANDLE();
    logger("hdc_BH_SetDCPenColor%s = get_specific_HANDLE(%d);", vid,
           hdc_BH_SetDCPenColor.index);
    cr_BH_SetDCPenColor = get_COLORREF(vid);
    logger("SetDCPenColor(hdc_BH_SetDCPenColor%s, cr%s);", vid, vid);
    SetDCPenColor(hdc_BH_SetDCPenColor.value, cr_BH_SetDCPenColor);
}
```

Case Study: Fuzzing Windows Helper Functions

MWR LABS

- Numerous structures defined in Windows.
  - RECT, POINT, SIZE FILETIME, etc.
- OS API calls act on these.
- Helper functions generate and populate these structures.

helpers.h

```
MENUITEMINFO get_MENUITEMINFO(char *vid);
LOGBRUSH get_LOGBRUSH(char *vid);
BITMAPINFOHEADER get_BITMAPINFOHEADER(char *vid);
COLORREF get_COLORREF(char *vid);
// …
POINT get_POINT(char *vid);
BLENDFUNCTION get_BLENDFUNCTION(char *vid);
```

Case Study: Fuzzing Windows Helper Functions

MWR LABS

MSDN

```
typedef DWORD COLORREF;
```

helpers.h

```
COLORREF get_COLORREF(char *vid) {

    COLORREF cr;
    logger("COLORREF cr%s;", vid);

    cr = get_fuzzed_uint32();

    logger("cr%s = %d;", vid, cr);
    return cr;
}
```

Case Study: Fuzzing Windows Helper Functions

MWR LABS

MSDN

```
typedef DWORD COLORREF;
```

helpers.h

```
COLORREF get_COLORREF(char *vid) {

    COLORREF cr;
    logger("COLORREF cr%s;", vid);

    cr = get_fuzzed_uint32();

    logger("cr%s = %d;", vid, cr);
    return cr;
}
```

MSDN

```
typedef struct _RECT {
    LONG left;
    LONG top;
    LONG right;
    LONG bottom;
} RECT, *PRECT;
```

helpers.h

```
RECT get_RECT(char *vid) {
    RECT rct;
    logger("RECT rct%s;", vid);
    rct.left = get_fuzzed_int32();
    rct.top = get_fuzzed_int32();
    rct.right = get_fuzzed_int32();
    rct.bottom = get_fuzzed_int32();
    logger("rct%s.left = %d;", vid, rct.left);
    logger("rct%s.top = %d;", vid, rct.top);
    logger("rct%s.right = %d;", vid, rct.right);
    logger("rct%s.bottom = %d;", vid, rct.bottom);
    return rct;
}
```

MWR LABS

MSDN

```
typedef struct _RECT {
    LONG left;
    LONG top;
    LONG right;
    LONG bottom;
} RECT, *PRECT;
```

helpers.h

```
RECT get_RECT(char *vid) {
    RECT rct;
    logger("RECT rct%s;", vid);
    rct.left = get_fuzzed_int32();
    rct.top = get_fuzzed_int32();
    rct.right = get_fuzzed_int32();
    rct.bottom = get_fuzzed_int32();
    logger("rct%s.left = %d;", vid, rct.left);
    logger("rct%s.top = %d;", vid, rct.top);
    logger("rct%s.right = %d;", vid, rct.right);
    logger("rct%s.bottom = %d;", vid, rct.bottom);
    return rct;
}
```

Case Study: Fuzzing Windows Worker Bootstrap

MWR LABS

- Assume fresh Windows with Python 3.5 installed.
- The worker bootstrapping script will set up the rest.
  - Installs Windows Debugger.
  - Installs Python's CouchDB module.
  - Performs minor Windows Registry tweaks.
  - Enables Windows kernel memory dumps.
  - Enables Special Pool for WIN32K.SYS.
  - Schedules the fuzzer control script to start on logon.
  - Reboots the system.

Case Study: Fuzzing Windows Crash Handling

MWR LABS

- Fuzzer kicks in on logon.
  - Checks for a kernel memory dump.
  - Creates a WinDbg log from running '!analyze' using 'kd.exe'.
  - Checks for a leftover fuzzer log.
- Bundles memory dump, WinDbg log, and fuzzer log.
- Submits to a central CouchDB instance.

MWR LABS

Other Operating Systems

- Proof of Concepts
  - Mac OS X
  - QNX
- System Calls
  - System V AMD64 ABI
- Object Store
  - File Descriptors
- Crash Detection
  - Kernel Debugging
  - System Logs

MWR LABS

```
void BH_IOBSDNameMatching() {

    CFMutableDictionaryRef returnValue_IOBSDNameMatching;
    int options_IOBSDNameMatching;
    char * bsdName_IOBSDNameMatching;

    char vid[16];
    sprintf(vid, "%d%d", get_time_in_ms(), rand() % 1024);

    logger("CFMutableDictionaryRef returnValue_IOBSDNameMatching%s;", vid);
    logger("int options_IOBSDNameMatching%s;", vid);
    logger("char * bsdName_IOBSDNameMatching%s;", vid);

    options_IOBSDNameMatching = get_fuzzed_int32();
    logger("options_IOBSDNameMatching%s = %d;", vid, options_IOBSDNameMatching);
    // …
```

```
    // …
    if(rand()) {
        bsdName_IOBSDNameMatching = "en0";
        logger("bsdName_IOBSDNameMatching%s = \"en0\";", vid);
    }
    else {
        bsdName_IOBSDNameMatching = "disk0s2"
        logger("bsdName_IOBSDNameMatching%s = \"disk0s2\";", vid);
    }

    logger("returnValue_IOBSDNameMatching%s =
IOBSDNameMatching(kIOMasterPortDefault, options_IOBSDNameMatching%s,
bsdName_IOBSDNameMatching%s);", vid, vid, vid);
    returnValue_IOBSDNameMatching = IOBSDNameMatching(kIOMasterPortDefault,
options_IOBSDNameMatching, bsdName_IOBSDNameMatching);

}
```

MWR LABS

http://opensource.apple.com/source/xnu/xnu-3248.50.21/bsd/kern/syscalls.master

```
0       AUE_NULL        ALL     { int nosys(void); }   { indirect syscall }
1       AUE_EXIT        ALL     { void exit(int rval) NO_SYSCALL_STUB; }
2       AUE_FORK        ALL     { int fork(void) NO_SYSCALL_STUB; }
3       AUE_NULL        ALL     { user_ssize_t read(int fd, user_addr_t cbuf,
user_size_t nbyte); }
4       AUE_NULL        ALL     { user_ssize_t write(int fd, user_addr_t cbuf,
user_size_t nbyte); }
5       AUE_OPEN_RWTCALL        { int open(user_addr_t path, int flags, int mode)
NO_SYSCALL_STUB; }
6       AUE_CLOSE       ALL     { int close(int fd); }
7       AUE_WAIT4       ALL     { int wait4(int pid, user_addr_t status, int options,
user_addr_t rusage) NO_SYSCALL_STUB; }
8       AUE_NULL        ALL     { int enosys(void); }   { old creat }
9       AUE_LINK        ALL     { int link(user_addr_t path, user_addr_t link); }
10      AUE_UNLINK      ALL     { int unlink(user_addr_t path) NO_SYSCALL_STUB; }
11      AUE_NULL        ALL     { int enosys(void); }   { old execv }
// …
```

MWR LABS

Results

- Windows
  - Numerous crashes in WIN32K.SYS.
- Other Operating Systems
- Hypervisors
  - VMWare Workstation crashes.
  - Several bugs in VMWare Tools, Virtual Box Guest Additions.

MWR LABS

Results: Windows

Host OS	Windows 10 Professional 64bit
Hypervisor	VMWare Workstation 12.1.0
Guests OS	Windows 7 Home Basic 64-bit
VM Specification	2 GB RAM and 1 CPU

MWR LABS

Results: Windows

Number of VMs	16
Hours	48
Total Number of Crashes	65
Unique Crashes	13

MWR LABS

Results: Windows

Null Pointer Dereference	4
Use-After-Free	2
Pool Buffer Overflow	4
Miscellaneous	3

MWR LABS

Results: Windows

DRIVER_PAGE_FAULT_BEYOND_END_OF_ALLOCATION (D6)	3
DRIVER_PAGE_FAULT_IN_FREED_SPECIAL_POOL (D5)	2
IRQL_NOT_LESS_OR_EQUAL (A)	1
KMODE_EXCEPTION_NOT_HANDLED (1E)	1
SPECIAL_POOL_DETECTED_MEMORY_CORRUPTION (C1)	1
SYSTEM_SERVICE_EXCEPTION (3B)	5

MWR LABS

Further Work

- Increase Coverage
  - Object Tagging
  - Implementing More Calls
  - Experimental User-Mode Callbacks
  - Better Multithreading Support
  - Coverage Feedback Based On CPU Features
- Miscellaneous
  - Logging
  - Handling Hypervisor Crashes
  - Test Cases Reducer
  - Monitoring VM Load and Reboot from Hypervisor

MWR LABS

Acknowledgements

- MWR Labs
- Nils
- Alex Plaskett
- Yong Chuan Koh
- Andrew Howe

MWR LABS

Feedback

Source code will shortly be available on GitHub.

@NerdKernel

@munmap

@MWRLabs