Dynamic Binary Instrumentation (DBI)

October 2025

cisco | TALOS

Who am I ?

Technical leader - Security Researcher at Cisco Talos

Malware Analysis, Threat hunting, Tool development, Machine Learning
Won the IDA plugin contest 2020 with the Dynamic Data Resolver Instrumentation Plugin

Germany

Holger Unterbrink

@hunterbr72

Agenda

This presentation is meant to be a workshop about:

- What is binary instrumentation ?
- DynamoRio Overview
- Practical examples of how to build your own instrumentation tools
  - Code
  - Build Environment
  - Caveats

Binary Instrumentation (BI)

What is binary instrumentation ?

Binary instrumentation is the process of inserting code into a compiled executable — without changing its source code — to monitor, analyze, or modify its behavior at runtime or before execution.

It's widely used for:

- Profiling and performance analysis
- Security auditing (e.g. detecting buffer overflows, fuzzing, etc)
- Debugging and tracing
- Program analysis and reverse engineering

Static Binary Instrumentation (SBI)

- Injects code before the binary is run (modifies the binary on disk).
- Tools: BAP, Dyninst (can do static), some LLVM passes

Dynamic Binary Instrumentation (DBI)

- Injects code at runtime, while the program is executing.
- Intercepts execution at the instruction level.
- Tools like: DynamoRIO, Intel PIN, Valgrind, Frida, QDBI

Feature	DynamoRIO	Intel PIN	Frida
Type	Dynamic Binary Instrumentation (DBI)	Dynamic Binary Instrumentation (DBI)	Dynamic Runtime Instrumentation via API Hooking
Instrumentation Granularity	Basic Blocks & Instructions	Instruction-level (very fine-grained)	Function-level & Instruction-level (via memory hooks)
Language (API)	C / C++	C / C++	JavaScript, Python, C
Target Platforms	Windows, Linux (limited macOS, Android forks)	Windows, Linux (x86/x64 only)	Windows, Linux, macOS, Android, iOS
Architecture Support	x86, x64, ARM (partial), AArch64 (forks)	x86, x64	x86, x64, ARM, ARM64
License	**Open Source (BSD-like)**	**Proprietary (Free for non-commercial)**	Open Source Core + Commercial License (Frida Pro)
Performance Overhead	**Medium (2–10× depending on tool complexity)**	High (10–20× or more with deep instrumentation)*	High (especially with many hooks or on mobile)
Transparency (Anti-Debug Evasion)	**Medium (code caching may leak)**	Medium to Low (can be fingerprinted)	Low (easily detectable by injected libraries or syscalls)
Best Use Cases	Runtime analysis, instrumentation, sandboxing	Deep instruction analysis, academic research	API hooking, mobile analysis, debugging, live patching
Shellcode Detection Feasibility	Excellent (module-level execution monitoring)	Good, but more effort needed	Limited (good for allocation + hook, not raw exec detection)
Community / Documentation	Active community, used in research + industry	Older, still maintained by Intel	Very active, large community, modern docs

Why DBI ?

Bypassing Anti-Analyzing Techniques

- Fight VM detection / Run on bare metal
- Fight Anti Debug
- Fight Anti Emulation
- Fight Anti Tamper
- Fight Anti Disasm
- Fight Obfuscation e.g. code traces
- Find interesting behavior e.g. shellcode execution
- Dumping memory and gather runtime information
- Automation

Code Analyser

Calculate all dynamic values used in indirect addressing

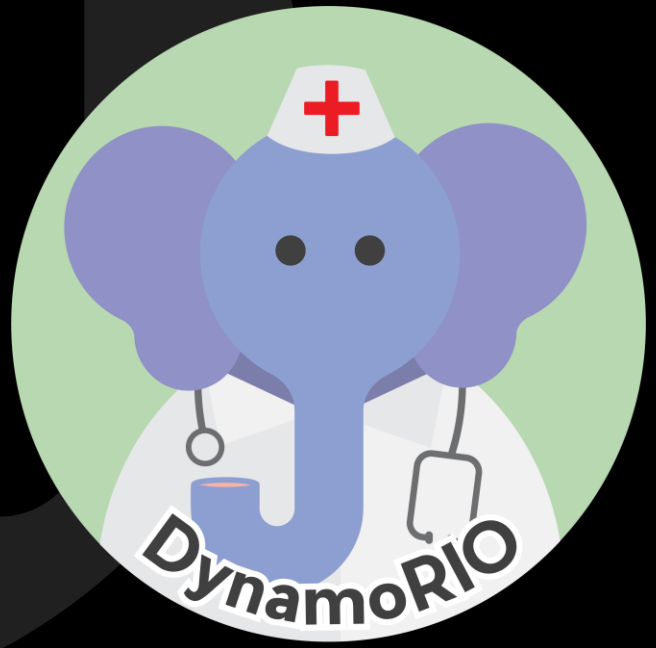
```
sar     eax, 0x9b                                   ; r15=0x0  eax:0xe7
mov     ecx, 0x0582c6bc                             ; eax=0x0  ecx:0x42c800
call    0x0000000140002a195                         ; ecx=0x582c6bc
mov     r14, qword ptr [rsp+r14-0xdf]               ; r14:0xe7  rsp:0x14fe60  r14:0xe7  [0x14fe60 + 0xe7 - 0xdf =
mov     rsi, qword ptr [rsp+rax+0x10]               ; r14=0x0  rsi:0x14fdb0  rsp:0x14fe60  rax:0x0  [0x14fe60 +
cmovp   di, cx                                      ; rsi=0x0  di:0x0  cx:0xc6bc
xor     eax, ecx                                    ; di=0xc6bc  eax:0x0  ecx:0x582c6bc
mov     rax, qword ptr [rsp+rax-0x0582c6a4]         ; eax=0x582c6bc  rax:0x582c6bc  rsp:0x14fe60  rax:0x582c6bc
lea     r10, [rcx+0x2183f321]                       ; rax=0xdead133c  r10:0x2d04d7d6  rcx:0x582c6bc  [0x582c6bc
pop     rbp                                         ; r10=0x2706b9dd  rbp:0x117ce
lea     rbp, [rdi*2-0x127161e8]                     ; rbp=0x1400256a6  rbp:0x1400256a6  rdi:0xe71c00000000c6bc
mov     r10, qword ptr [rsp+rcx-0x0582c6a4]         ; rbp=0xce37ffffed902b90  r10:0x2706b9dd  rsp:0x14fe68  rcx:
mov     rbp, qword ptr [rsp+rcx*2-0x0b058d58]       ; r10=0x12  rbp:0xce37ffffed902b90  rsp:0x14fe68  rcx:0x582c
mov     rdi, qword ptr [rsp+rcx-0x0582c694]         ; rbp=0x14fee0  rdi:0xe71c00000000c6bc  rsp:0x14fe68  rcx:0x
mov     qword ptr [rsp+rcx-0x0582c69c], 0x393d7487; rdi=0x4410b0  rsp:0x14fe68  rcx:0x582c6bc  [0x14fe68 + 0x
dec     qword ptr [rsp+rcx*2-0x0b058d58]            ; rsp:0x14fe68  rcx:0x582c6bc  [0x14fe68 + 0xb058d78 - 0xb058
add     cl, byte ptr [rsp+rcx*8-0x2c1635bc]         ; cl:0xbc  rsp:0x14fe68  rcx:0x582c6bc  [0x14fe68 + 0x2c1635
push    qword ptr [rsp+0x30]                        ; cl=0xbc  rsp:0x14fe68  [0x14fe68 + 0x30 = 0x14fe98]->0x000
```

```
r15=0x0  eax:0xe7
eax=0x0  ecx:0x42c800
ecx=0x582c6bc
r14:0xe7  rsp:0x14fe60  r14:0xe7  [0x14fe60 + 0xe7 - 0xdf = 0x14fe68]->0x0000000000000
r14=0x0  rsi:0x14fdb0  rsp:0x14fe60  rax:0x0  [0x14fe60 + 0x0 + 0x10 = 0x14fe70]->0x0
```

DynamoRio

Basics

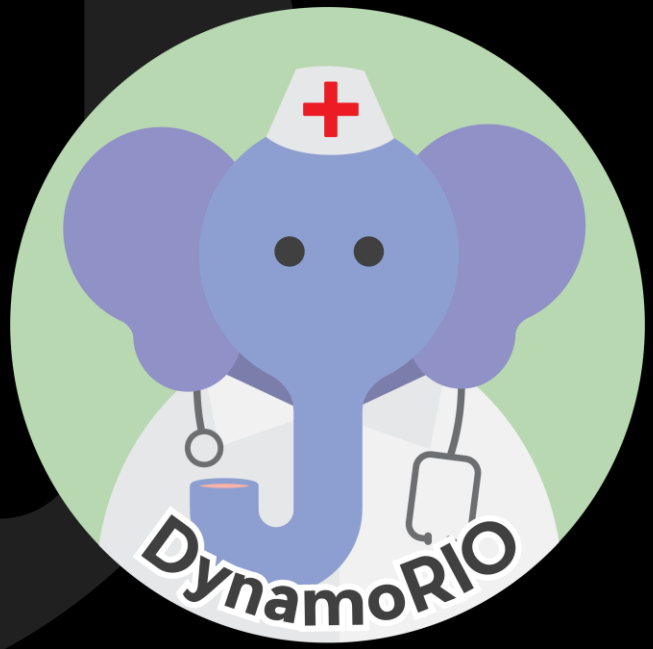
DynamoRio
https://dynamorio.org/



- Process Virtual Machine
- Multi platform support (x86/x64/ARM..)
- Multi OS support (Linux, Windows, Mac,…)
- Runtime code manipulation system
  - Runtime monitoring
  - Runtime patching
- History: Collaboration between MIT and Hewlett-Packard started in 2001
- MIT -> Determina -> VMware -> Google
- Open Source since February 2009

https://dynamorio.org/page_download.html

DynamoRio
https://dynamorio.org/



DynamoRIO includes tools for:

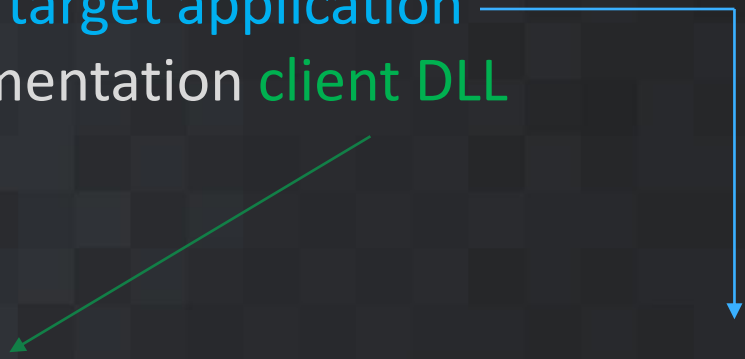
- **Profiling** (instruction count, memory traces, coverage)
- **Instrumentation** (basic blocks, syscalls, etc.)
- **Cache simulation**
- **Custom tool development**
- Easy installation: Just unzip to a folder

https://dynamorio.org/page_download.html

DynamoRio Basics

The common way to instrument a target application

drrun.exe is the launcher tool in DynamoRIO. It runs your target application under DynamoRio's control, injecting your custom instrumentation client DLL into the process.



<DynRio-InstallDir>\bin64\drrun.exe -c “<SOMEDIR>\client.dll" -- “<SOMEDIR>\anti_x.exe”

client.dll = your code (DynamoRio Extension) which does something with the binary, e.g. patching, tracing the instructions, etc.

…drrun.exe -c <SOMEDIR>\client.dll –arg1 –arg2 –arg3 -- anti_x.exe”

client parameters

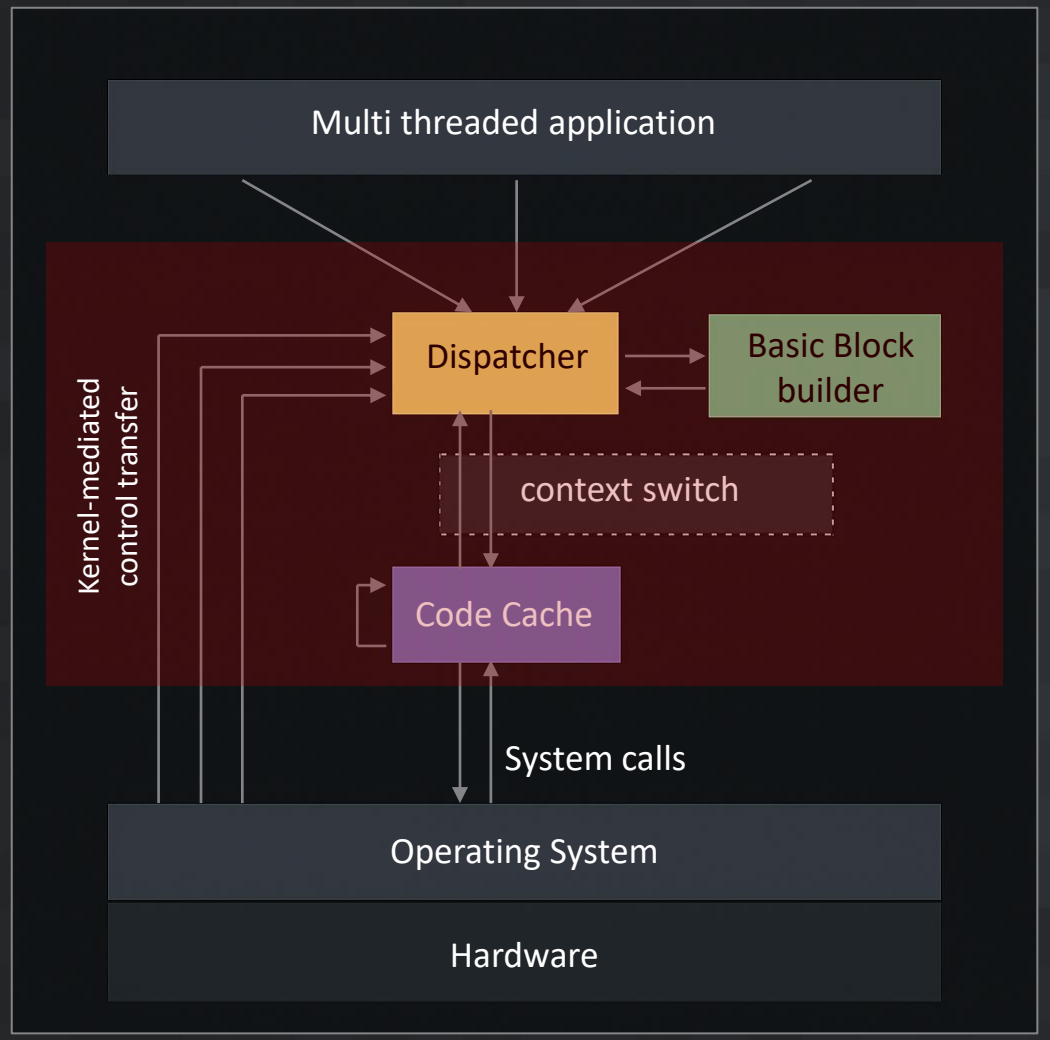
drrun.exe

Simplified Injection Flow

1. Starts the target app (e.g., sample.exe)
2. Injects the DynamoRIO core runtime into the process
3. Loads your client DLL (e.g., simple_client.dll)
4. Let your client observe and instrument code execution via callbacks like:
   - drmgr_register_bb_instrumentation_event()
   - drmgr_register_module_load_event()
   - drmgr_register_thread_init_event()
   - drmgr_register_thread_exit_event()
   - etc.
5. Executes the instrumented application as if it were running normally

DynamoRio Internals

DR is a Process Virtual Machine operating in user space



https://dynamorio.org/overview.html

Code Cache

- The code cache is a special memory region DynamoRio has full control over
- At runtime DynamoRio intercepts every basic block of the target application and copies it to the code cache
- As far as it has full control DynamoRio can modify the original instructions inside the code cache.
- DR taking over whenever control (context switch) leaves the code cache or when the operating system directly transfers control to the application

Code Cache Example

Simplified

Org. application code

Basic Block (BB)

```
loop_start:
  mov eax, ebx
  add eax, 1
  cmp eax, 10
  jne loop_start
```

Copied to code cache
incl. patches

Code cache

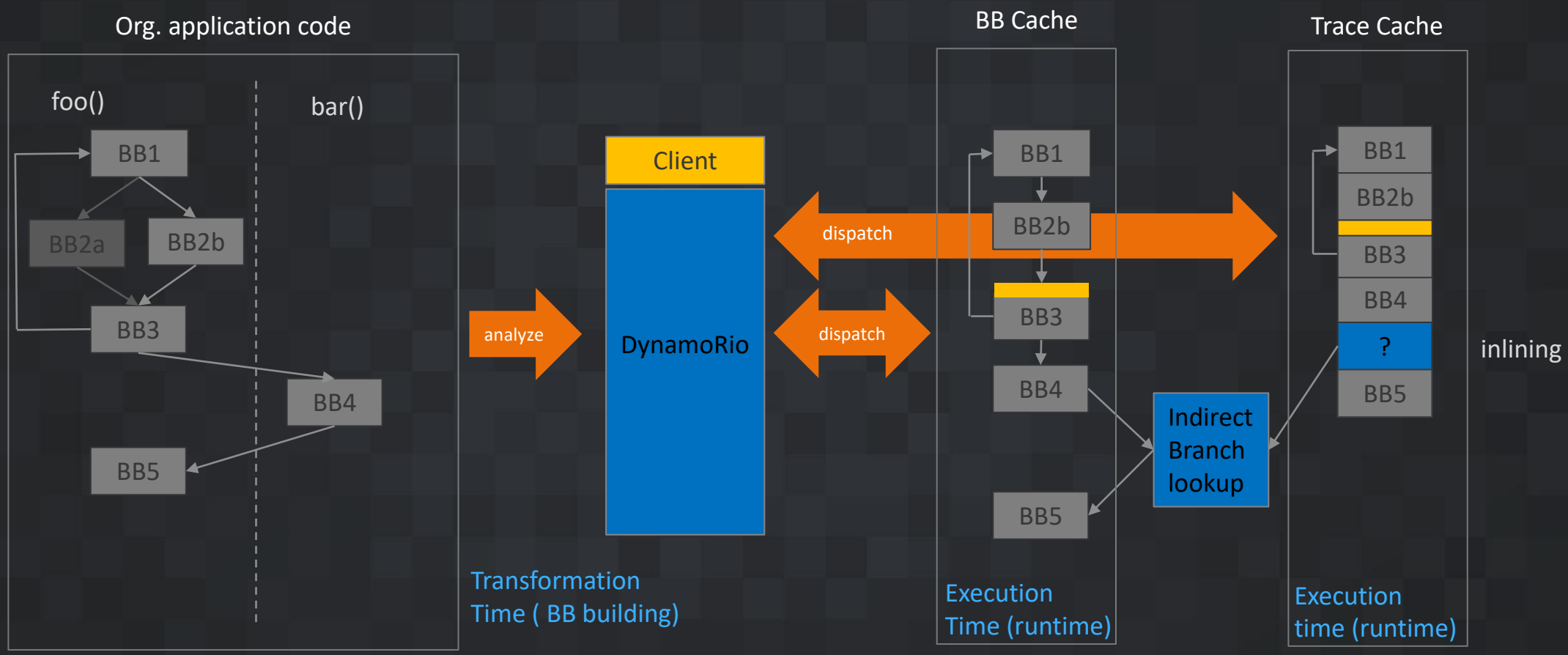
```
loop_start_in_bb_cache :
  jmp SomePatch
  mov eax, ebx
  add eax, 2
  cmp eax, 10
  jne loop_start_in_bb_cache
```

1. Decodes the basic block starting at loop_start
2. Stores a copy in the Basic Block Cache
3. Patches the jne instruction to jump to the cached loop_start if taken
4. On future iterations, it jumps straight to the cached version
5. At the end of the block (e.g., a jump), control is handed back to DynamoRIO's dispatcher.

Code Cache Details

BB Cache optimization with code trace cache

The code path is not yet considered *hot* (i.e., not frequently executed).



The **first time** a block of code is encountered, it is decoded and stored in the Basic Block Cache.

When a path is taken often enough, it's recorded as a trace

https://dynamorio.org/overview.html#sec_intro

Transparency

DR tries to avoid changing the behavior of the instrumented program.

Register State	DynamoRIO saves and restores all registers it modifies so that the application sees the original values.
Stack and Memory	Internal data used by DynamoRIO (e.g., for managing its own state) is hidden from the application.
Instruction Semantics	A transformed or instrumented instruction behaves identically to the original — no change in meaning or side effects.
Control Flow	Even if DynamoRIO adds trampolines or modifies jumps, the logical flow of execution remains the same from the program's point of view.
Signals and Exceptions	Any exceptions (e.g., SIGSEGV) or signals are handled in a way that preserves original behavior and context.

It should appear to the program as if it were running natively on the hardware — unaware of and unaffected by any instrumentation.

Hands on

Let’s write a client !

Development Environment

- DR supports Windows, Linux, Android, plus experimental MacOS support. Supported hardware is IA-32,AMD64,ARM, AArch64.
- All Examples in this Presentation are written with DynamoRIO 11.3.0

- Development Environment used:
  - Windows 11 Version 24H2 (Default Settings)*
  - Install Visual Studio 2019*, CMAKE (incl. in VS)
    Install [Strawberry] Perl
  - Download and unzip DynamoRio 11.3.0 from
    https://dynamorio.org/page_releases.html

Official Docs: https://dynamorio.org/page_build_client.html
If you need help: http://groups.google.com/group/dynamorio-users/

DynamoRio Client hello_world

Most simple client DLL

```
$ ccat client.c
#include "dr_api.h"

DR_EXPORT void
dr_client_main(client_id_t id, int argc, const char* argv[]) {

    // Enable console output
    dr_enable_console_printing();

    // Say hello
    dr_printf("Hello from DynamoRIO client!\n");
}
```

DynamoRio Client hello_world

Most simple CMake file

```
$ ccat CMakeLists.txt
cmake_minimum_required(VERSION 3.14)
project(simple_client C CXX)

# Add this to let CMake find the DynamoRIOConfig.cmake
list(APPEND CMAKE_PREFIX_PATH "C:/tools/DynamoRIO-Windows-11.3.0/cmake")

# Find the DynamoRIO package (uses the config file in the path above)
find_package(DynamoRIO REQUIRED CONFIG)

# Define the client as a shared library
add_library(simple_client SHARED client.c)

# Mark this as a DynamoRIO client (sets flags, includes, etc.)
configure_DynamoRIO_client(simple_client)

# Optionally install the resulting DLL somewhere
install(TARGETS simple_client DESTINATION bin)
```

Building the client library

CMAKE build process

Building the client

1. Start Developer Prompt 2019 and change to your clients directory
2. cmake -S . -B build -G "Visual Studio 16 2019" -A x64 -DCMAKE_BUILD_TYPE=Release
3. cmake --build build --config Release --verbose

Optional, in case you want to see all gory details like compiler or linker flags

Other Options: RelWithDebInfo or Debug

Execute and test

<DynRio-InstallDir>\bin64\drrun.exe -c "<*SOMEDIR*>\simple_client.dll" – "<*SOMEDIR*>\sample.exe"
[-disable_traces] # optional for disabling the trace cache optimization

DynamoRio Client hello_world - Example

```
C:\simple_client>cmake -S . -B build -G "Visual Studio 16 2019" -A x64 -DCMAKE_BUILD_TYPE=Release
-- Selecting Windows SDK version 10.0.26100.0 to target Windows 6.2.9200.
-- The C compiler identification is MSVC 19.29.30159.0
-- The CXX compiler identification is MSVC 19.29.30159.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Professional/VC/Tools/MSVC/14.29.30133/bin/Hostx64/x64/cl.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Professional/VC/Tools/MSVC/14.29.30133/bin/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Configuring done (4.9s)
-- Generating done (0.0s)
-- Build files have been written to: C:/Users/hunte/source/repos/simple_client/build

C:\simple_client>cmake --build build --config Release --verbose

> cd C:\simple_client\build\Release

> C:\tools\DynamoRIO-Windows-11.3.0\bin64\drrun.exe -c simple_client.dll -- notepad.exe

Hello from DynamoRIO client!
```

Yay, we wrote our first instrumentation client !

A real world client

But still simple… something which actually does something useful

DynamoRio Client CMakeLists.txt

Building a simple Client DLL which actually does something

```
# Init project
cmake_minimum_required(VERSION 3.15)
project(myclient LANGUAGES C CXX)

# Path variable to your DynamoRIO folder [optional]
set(DR_BASE "C:/tools/DynamoRIO-Windows-11.3.0")

# CMake needs to know where to find the DynamoRIOConfig.cmake
list(APPEND CMAKE_PREFIX_PATH "${DR_BASE}/cmake")

# Find the DynamoRIO package
find_package(DynamoRIO REQUIRED CONFIG)
```

DynamoRio Client CMakeLists.txt

Building a simple Client DLL which actually does something

```
# Add DynamoRIO include directories and define your client target
include_directories(${DynamoRIO_INCLUDE_DIRS})

# Define the client as a shared library
add_library(myclient SHARED myclient.c)

# Set the DLL name to "my_custom_client.dll"
set_target_properties(myclient PROPERTIES OUTPUT_NAME "simple_client")

# Mark this as a DynamoRIO client (sets flags, includes, etc.)
configure_DynamoRIO_client(myclient)
```

DynamoRio Client CMakeLists.txt

Building a simple Client DLL which actually does something

```
# Add the DR extensions which you are using to the build process e.g. drmgr , drwrap , drsyms…
use_DynamoRIO_extension(myclient drmgr)          # multi-instrumentation mediation
use_DynamoRIO_extension(myclient drwrap)         # function wrapping and replacing
…

# Link the DynamoRIO library to your client.
target_link_libraries(myclient PRIVATE ${DynamoRIO_LIBRARIES})
```

DynamoRio Client CMakeLists.txt

Building a simple Client DLL which actually does something

```
# Compiler settings for MSVS [optional]
target_compile_definitions(myclient PRIVATE WINDOWS X86_64)  # Use X86_32 for 32-bit targets
if(MSVC)
    # Enable higher warning level and multi-processor compilation
    target_compile_options(myclient PRIVATE /W4 /MP)
endif()

# Suppress "LNK4281: undesirable base address 0x72000000 for x64 image" warning
# DynamoRio uses set(preferred_base "0x71000000") in its CMakeLists.txt file to
# keep its client DLL close to its own memory regions [optional]
target_link_options(myclient PRIVATE "/IGNORE:4281")
```

DynamoRio Client CMakeLists.txt

Building a simple Client DLL which actually does something

```
# [optional] For debugging the build process
get_target_property(LINK_LIBS myclient LINK_LIBRARIES)
message(STATUS "myclient Linked Libraries: ${LINK_LIBS}")


# [optional] Install the resulting DLL somewhere
install(TARGETS myclient DESTINATION bin)
```

DR Extentions

Check the DR docs for details

- drreg: register stealing and allocating
- drsyms: symbol table and debug information lookup
- drcontainers: hashtable, vector, and table
- drmgr: multi-instrumentation mediation
- drwrap: function wrapping and replacing
- drutil: memory tracing, string loop expansion
- drx: multi-process management, misc utilities
- drsyscall: system call monitoring
- drdecode: CPU decoding/encoding library
- umbra: shadow memory framework

DynamoRIO Extension Manager (drmgr)

Building a client for instrumentation

A helper library designed to make writing DynamoRIO clients easier, cleaner, and safer.

• Event Registration	Simplifies registering for events like bb_event, thread_init, etc.
• Thread-local Storage (TLS)	Savely manages TLS slots
• Per-thread cleanup	Automatically frees thread-specific memory on exit
• Safe initialization	Ensures extensions are initialized in the right order
• Avoids conflicts	Handles shared slot allocation so multiple extensions don't collide

For example:

drmgr_register_module_load_event(event_module_load_callback_func) // executed on module load e.g. DLL

```
20   # Add this line to include and link drmgr
21   use_DynamoRIO_extension(simple_client2 drmgr)
```

CMakeLists.txt

Docs:
https://dynamorio.org/group__drmgr.html

Event Registration

drmgr_register_bb_instrumentation_event()

```
DR_EXPORT bool drmgr_register_bb_instrumentation_event ( drmgr_analysis_cb_t  analysis_func,
                                                         drmgr_insertion_cb_t insertion_func,
                                                         drmgr_priority_t *   priority
                                                       )
```

Registers callback functions for the second and third instrumentation stages: application analysis and instrumentation insertion. drmgr will call `func` as the second of five instrumentation stages for each dynamic application basic block.

The first stage performed any changes to the original application code, and later stages are not allowed to change application code. Application analysis passes in the second stage are not allowed to add to or change the instruction list other than adding label instructions, and are intended for analysis of application code either for immediate use or for use by the third stage. Label instructions can be used to store data for use in subsequent stages with custom tags inserted as notes via **drmgr_reserve_note_range()** and custom data stored via **instr_get_label_data_area()**.

The third instrumentation stage is instrumentation insertion. Unlike the other stages, this one passes only one instruction to the callback, allowing each registered component to act on one instruction before moving to the next instruction. Instrumentation insertion passes are allowed to insert meta instructions only immediately prior to the passed-in instruction: not before any prior non-meta instruction nor after any subsequent non-meta instruction. They are not allowed to insert

```
// Register event callback functions
dr_printf("[TRACER] [DEBUG] [dr_client_main] Register callback functions\n");
drmgr_register_module_load_event(event_module_load_trace_instr);              // executed on module load e.g. DLL
//drmgr_register_thread_init_event(event_thread_init_trace_instr);            // executed on thread initalisation
drmgr_register_bb_instrumentation_event(NULL, event_bb_instr_global, NULL);   // executed on every basic block
//drmgr_register_thread_exit_event(event_thread_exit_trace_instr);            // executed on thread exit
dr_register_exit_event(event_exit);                                           // executed on process exit
```

Docs:
https://dynamorio.org/using.html#sec_events
https://dynamorio.org/group__drmgr.html

Code

```
1
2
3  .... switching to Code Example.
```

Function Wrapping and Replacing (drwrap)

Patching binaries – Wrapping before (pre) and after (post) function execution

```
void wrap_post_function(void *wrapcxt, void *user_data)
{
    UNUSED(user_data);
    dr_printf("[SIMPLECLIENT] [DEBUG] [wrap_post_function] Setting function post wrap\n");
    drwrap_set_retval(wrapcxt, (void *)0);
    dr_printf("[SIMPLECLIENT] [DEBUG] [wrap_post_function] Return value set to FALSE\n");
}
```

```
// Patch a function:
app_pc func_addr = (app_pc)0x140001070; // for imported functions "use func_addr = (app_pc)dr_get_proc_address(info->handle, "I

if (func_addr >= info->start && func_addr < info->end) {
    bool success = drwrap_wrap(func_addr, NULL, wrap_post_function);
    if (success) {
        dr_printf("[SIMPLECLIENT] [DEBUG] [event_module_load_trace_instr] Successfully wrapped function at %p\n", func_addr);
    } else {
        dr_printf("[SIMPLECLIENT] [DEBUG] [event_module_load_trace_instr] Failed to wrap function at %p\n", func_addr);
    }
}
```

Setting Pre-Function to NULL = No Pre-Function

Docs:
https://dynamorio.org/group__drwrap.html

Anti-X Cases

… and uncommon code

```
// Shellcode for selfmod func
unsigned char encoded_code[] = {
    0x50 ^ KEY,                         // 1. push rax
    0x50 ^ KEY,                         // 2. push rax
    0x50 ^ KEY,                         // 3. push rax
    0x50 ^ KEY,                         // 4. push rax
    0x50 ^ KEY,                         // 5. push rax
    0x50 ^ KEY,                         // 6. push rax
    0x58 ^ KEY,                         // 1. pop rax
    0x58 ^ KEY,                         // 2. pop rax
    0x58 ^ KEY,                         // 3. pop rax
    0x58 ^ KEY,                         // 4. pop rax
    0x58 ^ KEY,                         // 5. pop rax
    0x58 ^ KEY,                         // 6. pop rax
    0xC3 ^ KEY                          // ret
};

void xor_decrypt(unsigned char* data, size_t len) {
    for (size_t i = 0; i < len; ++i)
        data[i] ^= KEY;
}

bool decode_and_run_shellcode() {
    SIZE_T code_size = sizeof(encoded_code);
    void* dec_shellcode = VirtualAlloc(NULL, code_size, MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
    if (!dec_shellcode) {
        printf("[ANTI-X] VirtualAlloc failed.\n");
        return 1;
    }

    memcpy(dec_shellcode, encoded_code, code_size);
    xor_decrypt((unsigned char*)dec_shellcode, code_size);

    printf("[ANTI-X] running decoded shellcode ...\n");
    ((void(*)())dec_shellcode)();

    return 0;
}
```

Shellcode

- Shellcode decoding and execution at runtime
- Usually no problem
- Dump Memory with dr_safe_read()

BTW dump memory … SSL/TLS Traffic

Works out of the box*

```
hConnect = WinHttpConnect(hSession, L"ifconfig.me",
    INTERNET_DEFAULT_HTTPS_PORT, 0);
if (!hConnect) {
    error = GetLastError();
    printf("[ANTI-X] [ERROR] WinHttpConnect failed with error
    goto cleanup;
}
```

```
if (get_external_ip(myip, 10000)) {
    printf("[ANTI-X] Successfully received data from server.\n");
}
else {
    printf("[ANTI-X] HTTPS request failed\n");
}
```

```
[SIMPLECLIENT] [DEBUG] [event_module_load_trace_instr] Process PID 14188
[ANTI-X] Successfully received data from server.
[ANTI-X] Done.
```

- Transparent
- Get C2 URL/Domains
- Intercept C2 traffic
- Manipulate C2 traffic
- You name it …

*TBD: 12175 ERROR_WINHTTP_SECURE_FAILURE

Code validation

```
// Function to calculate CRC32 checksum
uint32_t crc32(const void* data, size_t length) {
    uint32_t crc = 0xFFFFFFFF;
    const uint8_t* buf = (const uint8_t*)data;

    for (size_t i = 0; i < length; i++) {
        //printf("[ANTI-X] buf[i] = %p\n", &(buf[i]));
        crc ^= buf[i];
        for (int j = 0; j < 8; j++) {
            if (crc & 1)
                crc = (crc >> 1) ^ 0xEDB88320;
            else
                crc >>= 1;
        }
    }

    return ~crc;
}

void checkmem_crc32 {
    uint32_t crc = crc32(func_start, funcSize);

    if (crc == CRC32VALUE_OF_MYFUNC) {
        printf("[ANTI-X] [SUCCESS] CRC32 matches! Function code is ok.\n");
    }
    else {
        printf("[ANTI-X] [INTEGRITY CHECK FAIL] CRC32 does not match!\n");
        printf("[ANTI-X] [INTEGRITY CHECK FAIL] Function code is modified.\n");
        printf("[ANTI-X] [INTEGRITY CHECK FAIL] Function might be debugged or was mod
    }
}
```

- Memory validation checks
- Transparent in DynamoRio

Anti-X Debugger Checks

```
// Check for debugger presence (PEB->BeingDebugged)
if (IsDebuggerPresent()) {
    printf("[ANTI-X] [INTEGRITY CHECK FAIL] Process is debugged! PEB->BeingDebugged\n");
}
else {
    printf("[ANTI-X] [SUCCESS] No debugger detected.\n");
}
```

- Simple Debugger checks
- Transparent in DynamoRio

```
// Check for hardware breakpoints
HANDLE hThread = GetCurrentThread();
CONTEXT ctx;

ctx.ContextFlags = CONTEXT_DEBUG_REGISTERS;

if (GetThreadContext(hThread, &ctx)) {

    if (ctx.Dr0 != 0 || ctx.Dr1 != 0 || ctx.Dr2 != 0 || ctx.Dr3 != 0) {
        printf("[ANTI-X] [INTEGRITY CHECK FAIL] GetThreadContext: Hardware breakpoints detected!\n");
    }
    else {
        printf("[ANTI-X] [SUCCESS] GetThreadContext: No hardware breakpoints detected.\n");
    }
}
else {
    printf("[ANTI-X] [ERROR] GetThreadContext failed: %lu\n", GetLastError());
}
```

- Simple Debugger checks
- HW BP via Thread context

Anti-X Debugger Checks –DR Fail

```
// Check for hardware breakpoint again with a different trick  (this breaks DynamoRio)
CONTEXT* ctx2;
SIZE_T debugger_attached = 0;
__try {
    __writeeflags(__readeflags() | 0x100);  // Set TF flag aka set CPU to single step
    __nop();                                // trigger exception in single step mode
}
__except (ctx2 = (GetExceptionInformation())->ContextRecord,
    debugger_attached = (ctx2->ContextFlags & CONTEXT_DEBUG_REGISTERS) ?
    ctx2->Dr0 | ctx2->Dr1 | ctx2->Dr2 | ctx2->Dr3 : 0,
    EXCEPTION_EXECUTE_HANDLER)
{
    if (debugger_attached) {
        printf("[ANTI-X] [INTEGRITY CHECK FAIL] Exception test: Hardware breakpoints detected!\n");
    }
    else {
        printf("[ANTI-X] [SUCCESS] Exception test: No hardware breakpoints detected.\n");
    }
}
```

- DR just stops executing the target app with out any error msg

Anti-X

DR just exiting

Run without the previous shown SEH exception trick

```
[ANTI-X] [SUCCESS] GetThreadContext: No hardware breakpoints detected.
[ANTI-X] CPU-cycles: 231359
[ANTI-X] [SUCCESS] Runtime is ok!
[ANTI-X] This function always returns TRUE
[SIMPLECLIENT] [DEBUG] [wrap_post_function] Setting function post wrap
[SIMPLECLIENT] [DEBUG] [wrap_post_function] Return value set to FALSE
```

Run with the SEH exception trick

```
[ANTI-X] [SUCCESS] GetThreadContext: No hardware breakpoints detected.
[SIMPLECLIENT] [DEBUG] [event_exit] Number of instrumented instructions: 0

C:\Users\hunte\source\repos\simple_client4>
```

Anti-X – Common Exceptions

"Normal" exceptions are usually working fine

```
printf("[ANTI-X] triggering an exception...\n");
__try {
    char* p = NULL;
    *p = 0; // This will cause an access violation exception
    printf("[ANTI-X] This should never be reached due to the exception\n");
}
__except (EXCEPTION_EXECUTE_HANDLER) {
    DWORD code = GetExceptionCode();
    printf("[ANTI-X] Exception caught: %s (0x%08X)\n", DescribeException(code), code);
}
printf("[ANTI-X] Exception triggert.\n");
```



```
[ANTI-X] triggering an exception...
[ANTI-X] Exception caught: Access Violation (0xC0000005)
[ANTI-X] Exception triggert.
```

Anti-X – Runtime Checks

```
// Test runtime
uint64_t start_runtime, end_runtime, runtime;
int cpu_info[4];

__cpuid(cpu_info, 0);
start_runtime = __rdtsc();

test_function();

__cpuid(cpu_info, 0);
end_runtime = __rdtsc();

runtime = end_runtime - start_runtime;
printf("[ANTI-X] CPU-cycles: %I64u\n", runtime);

// Modify the hartcoded runtime value for your system
if (runtime < 1000000) {
    printf("[ANTI-X] [SUCCESS] Runtime is ok!\n");
}
else {
    printf("[ANTI-X] [INTEGRITY CHECK FAIL] Runtime is too long! Function might be debugged.\n");
}
```

- Works in most cases
- Depends on what you do
- Depends on how paranoid the check is

Anti-X

Be careful with instrumenting large loops

```
printf("[ANTI-X] Running a larger loop\n");
c = 0;
for (i = 0; i < 1000000000; i++) {
    if (i > 6) {
        c += i;
    }
}
```

- Just running it via drrun.exe is usually fine,
- …instrumenting it probably not.

DynamoRio is a high performance instrumentation tool, but any instrumentation comes with a price !

Anti-X – Self modifying code

```
1   selfmodify PROC
2       push    rbx             ; Save RBX
3   postpatch:
4       mov     rax, 1234h      ; <-- gets post-patched with 'mov rax, 1'
5       mov     rbx, 1234h      ; Load second value into RBX
6       cmp     rax, rbx        ; Compare RAX and RBX
7       je      equal_label     ; Jump if equal (ZF = 1)
8       mov     rax, 0h         ;
9       jmp     end_label       ;
10
11  equal_label:
12      mov     rax, 1h         ; rax = 1h
13      inc     rax             ; rax = 2h
14      call    somefunc1       ; rax = 666h
15      nop
16      lea     rsi, prepatch               ; Pre-Exec Patch
17      mov     byte ptr [rsi],   048h      ; 48 C7 C0 03 00 00 00 = 'mov rax, 3'
18      mov     byte ptr [rsi+1], 0C7h
19      mov     byte ptr [rsi+2], 0C0h
20      mov     byte ptr [rsi+3], 003h
21      mov     byte ptr [rsi+4], 000h
22      mov     byte ptr [rsi+5], 000h
23      mov     byte ptr [rsi+6], 000h
24      nop
25      nop
26      inc     rax                     ; rax = 4h
27      dec     rax                     ; rax = 3h
28      inc     rax                     ; rax = 4h
29      dec     rax                     ; rax = 3h
30  prepatch:
31      jmp     end_label               ; <-- gets pre-patched with: 'mov rax, 3'
32      nop                             ; 2nd run: rax = 0h
33      nop
34      nop
35      nop
36      nop
37      test    rax, rax                ; always set ZF = 0
38      jz      int_leav                ; Anti-Disassembler trick make a 'jmp,nop,nop' out of a 'mov'
39      jnz     int_leav+3              ; Anti-Disassembler trick make a 'jmp,nop,nop' out of a 'mov'
40  int_leav:
41      db      048h, 0C7h, 0C0h, 0ebh, 00Ch, 090h, 090h        ; eb 09 = jmp by 7bytes ('inc rax' two instr below)
42      mov     rbx, 0deadbeefh
43      inc     rax                                             ; jmp addr   rax = 1h
44      dec     rax                                             ;            rax = 0h
45
46  end_label:
47      push    rsi                     ; Save RSI
48      lea     rsi, postpatch          ; Post-Exec Patch
49      mov     byte ptr [rsi],   048h  ; 48 C7 C0 00 00 00 00  mov     rax, 1
50      mov     byte ptr [rsi+1], 0C7h
51      mov     byte ptr [rsi+2], 0C0h
52      mov     byte ptr [rsi+3], 001h
53      mov     byte ptr [rsi+4], 000h
54      mov     byte ptr [rsi+5], 000h
55      mov     byte ptr [rsi+6], 000h
56
57      pop     rsi             ; Restore RSI
58      pop     rbx             ; Restore RBX
59      ret
60  selfmodify ENDP
```

- DynamoRio detects self modifying code and usually handles it well
- Block gets flushed and re-created
- Global variables can be tricky

```
[SIMPLECLIENT] [DEBUG] [event_bb_instr_global] instrumenting: 0x140001eb8
<writing to executable region.>
<self-modifying code.>
[SIMPLECLIENT] [DEBUG] [event_bb_instr_global] instrumenting: 0x140001e8f
```

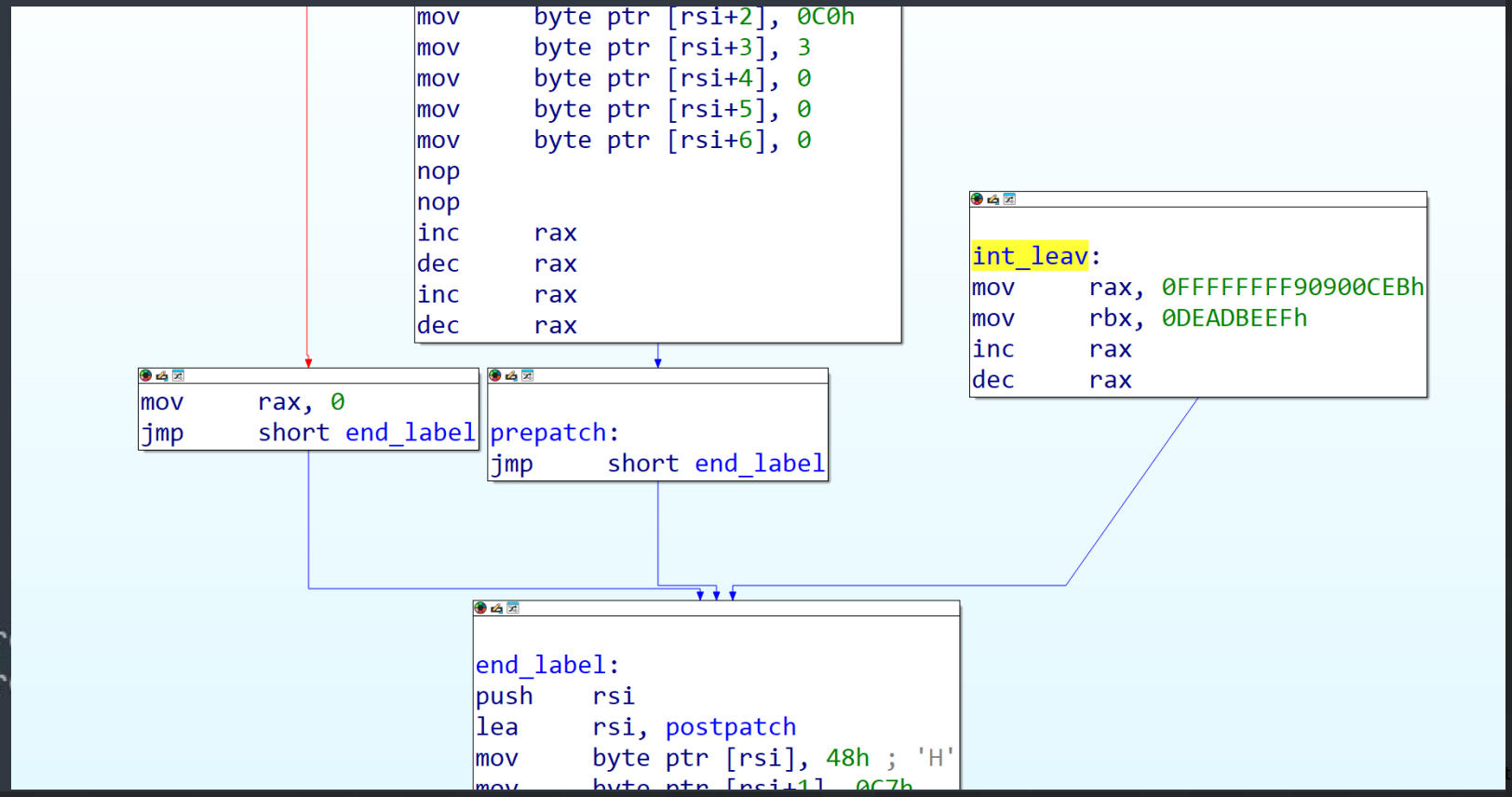
```
1   selfmodify PROC
2       push    rbx                 ; Save RBX
3   postpatch:
4       mov     rax, 1234h          ; <-- gets post-patched with 'mov rax, 1'
5       mov     rbx, 1234h          ; Load second value into RBX
6       cmp     rax, rbx            ; Compare RAX and RBX
7       je      equal_label         ; Jump if equal (ZF = 1)
8       mov     rax, 0h             ;
9       jmp     end_label           ;
10
11  equal_label:
12      mov     rax, 1h             ; rax = 1h
13      inc     rax                 ; rax = 2h
14      call    somefunc1           ; rax = 666h
15      nop
16      lea     rsi, prepatch                ; Pre-Exec Patch
17      mov     byte ptr [rsi],   048h       ; 48 C7 C0 03 00 00 00 = 'mov rax, 3'
18      mov     byte ptr [rsi+1], 0C7h
19      mov     byte ptr [rsi+2], 0C0h
20      mov     byte ptr [rsi+3], 003h
21      mov     byte ptr [rsi+4], 000h
22      mov     byte ptr [rsi+5], 000h
23      mov     byte ptr [rsi+6], 000h
24      nop
25      nop
26      inc     rax                  ; rax = 4h
27      dec     rax                  ; rax = 3h
28      inc     rax                  ; rax = 4h
29      dec     rax                  ; rax = 3h
30  prepatch:
31      jmp     end_label            ; <-- gets pre-patched with: 'mov rax, 3'
32      nop                          ; 2nd run: rax = 0h
33      nop
34      nop
```

- Change jmp to mov -> new control flow
- Change happens BEFORE the code location is executed (pre-patch)

```
30  prepatch:
31      jmp     end_label               ; <-- gets pre-patched with: 'mov rax, 3'
32      nop                             ; 2nd run: rax = 0h
33      nop
34      nop
35      nop
36      nop
37      test    rax, rax                ; always set ZF = 0
38      jz      int_leav                ; Anti-Disassembler trick make a 'jmp,nop,nop' out of a 'mov'
39      jnz     int_leav+3              ; Anti-Disassembler trick make a 'jmp,nop,nop' out of a 'mov'
40  int_leav:
41      db      048h, 0C7h, 0C0h, 0ebh, 00Ch, 090h, 090h        ; eb 09 = jmp by 7bytes ('inc rax' two instr below)
42      mov     rbx, 0deadbeefh
43      inc     rax                                             ; jmp addr   rax = 1h
44      dec     rax                                             ;            rax = 0h
45
46  end_label:
47      push    rsi                     ; Save RSI
48      lea     rsi, postpatch          ; Post-Exec Patch
49      mov     byte ptr [rsi],   048h  ; 48 C7 C0 00 00 00 00  mov     rax, 1
50      mov     byte ptr [rsi+1], 0C7h
51      mov     byte ptr [rsi+2], 0C0h
52      mov     byte ptr [rsi+3], 001h
53      mov     byte ptr [rsi+4], 000h
54      mov     byte ptr [rsi+5], 000h
55      mov     byte ptr [rsi+6], 000h
56
57      pop     rsi             ; Restore RSI
58      pop     rbx             ; Restore RBX
59      ret
60  selfmodify ENDP
```

```
30  prepatch:
31      jmp     end_label               ; <-- gets pre-patched with: 'mov rax, 3'
32      nop                             ; 2nd run: rax = 0h
33      nop
34      nop
35      nop
36      nop
37      test    rax, rax                ; always set ZF = 0
38      jz      int_leav                ; Anti-Disassembler trick make a 'jmp,nop,nop' out of a 'mov'
39      jnz     int_leav+3              ; Anti-Disassembler trick make a 'jmp,nop,nop' out of a 'mov'
40  int_leav:
41      db      048h, 0C7h, 0C0h, 0ebh, 00Ch, 090h, 090h        ;
42      mov     rbx, 0deadbeefh
43      inc     rax
44      dec     rax
45
46  end_label:
47      push    rsi
48      lea     rsi, postpatch
49      mov     byte ptr [rsi],   048h
50      mov     byte ptr [rsi+1], 0C7h
51      mov     byte ptr [rsi+2], 0C0h
52      mov     byte ptr [rsi+3], 001h
53      mov     byte ptr [rsi+4], 000h
54      mov     byte ptr [rsi+5], 000h
55      mov     byte ptr [rsi+6], 000h
56
57      pop     rsi                 ; Restor
58      pop     rbx                 ; Restor
59      ret
60  selfmodify ENDP
```

eb 0c = jmp 12 bytes down



```
30  prepatch:
31      jmp     end_label
32      nop
33      nop
34      nop
35      nop
36      nop
37      test    rax, rax
38      jz      int_leav
39      jnz     int_leav+3
40  int_leav:
41      db      048h, 0C7h, 0C0h, 0ebh, 00
42      mov     rbx, 0deadbeefh
43      inc     rax
44      dec     rax
45
46  end_label:
47      push    rsi
48      lea     rsi, postpatch
49      mov     byte ptr [rsi],   048h
50      mov     byte ptr [rsi+1], 0C7h
51      mov     byte ptr [rsi+2], 0C0h
52      mov     byte ptr [rsi+3], 001h
53      mov     byte ptr [rsi+4], 000h
54      mov     byte ptr [rsi+5], 000h
55      mov     byte ptr [rsi+6], 000h
56
57      pop     rsi             ; Restore
58      pop     rbx             ; Restore
59      ret
60  selfmodify ENDP
```

```
0140001EBF 48 85 C0                     test    rax, rax
0140001EC2 74 02                        jz      short near ptr int_
0140001EC4 75 03                        jnz     short loc_140001EC9
0140001EC4                      ; ---------------------------------------
0140001EC6 48                   int_leav        db  48h ; H             ; C
0140001EC7 C7                                   db 0C7h
0140001EC8 C0                                   db 0C0h
0140001EC9                      ; ---------------------------------------
0140001EC9
0140001EC9                      loc_140001EC9:                          ; C
0140001EC9 EB 0C                                jmp     short loc_140001ED7
0140001EC9                      ; ---------------------------------------
0140001ECB 90                                   db  90h
0140001ECC 90                                   db  90h
0140001ECD 48                                   db  48h ; H
0140001ECE BB                                   db 0BBh
0140001ECF EF                                   db 0EFh
0140001ED0 BE                                   db 0BEh
0140001ED1 AD                                   db 0ADh
0140001ED2 DE                                   db 0DEh
0140001ED3 00                                   db    0
0140001ED4 00                                   db    0
0140001ED5 00                                   db    0
0140001ED6 00                                   db    0
0140001ED7                      ; ---------------------------------------
0140001ED7
0140001ED7                      loc_140001ED7:                          ; C
0140001ED7 48 FF C0                             inc     rax
0140001EDA 48 FF C8                             dec     rax
```

```
30  prepatch:
31      jmp     end_label               ; <-- gets pre-patched with: 'mov rax, 3'
32      nop                             ; 2nd run: rax = 0h
33      nop
34      nop
35      nop
36      nop
37      test    rax, rax                ; always set ZF = 0
38      jz      int_leav                ; Anti-Disassembler trick make a 'jmp,nop,nop' out of a 'mov'
39      jnz     int_leav+3              ; Anti-Disassembler trick make a 'jmp,nop,nop' out of a 'mov'
40  int_leav:
41      db      048h, 0C7h, 0C0h, 0ebh, 00Ch, 090h, 090h        ; eb 09 = jmp by 7bytes ('inc rax' two instr below)
42      mov     rbx, 0deadbeefh
43      inc     rax                                             ; jmp addr   rax = 1h
44      dec     rax                                             ;            rax = 0h
45
46  end_label:
47      push    rsi                     ; Save RSI
48      lea     rsi, postpatch          ; Post-Exec Patch
49      mov     byte ptr [rsi],   048h  ; 48 C7 C0 00 00 00 00  mov     rax, 1
50      mov     byte ptr [rsi+1], 0C7h
51      mov     byte ptr [rsi+2], 0C0h
52      mov     byte ptr [rsi+3], 001h
53      mov     byte ptr [rsi+4], 000h
54      mov     byte ptr [rsi+5], 000h
55      mov     byte ptr [rsi+6], 000h
56
57      pop     rsi             ; Restore RSI
58      pop     rbx             ; Restore RBX
59      ret
60  selfmodify ENDP
```

```
1   selfmodify PROC
2       push    rbx             ; Save RBX
3   postpatch:
4       mov     rax, 1234h      ; <-- gets post-patched with 'mov rax, 1'
5       mov     rbx, 1234h      ; Load second value into RBX
6       cmp     rax, rbx        ; Compare RAX and RBX
7       je      equal_label     ; Jump if equal (ZF = 1)
8       mov     rax, 0h         ;
9       jmp     end_label       ;
```

Anti-X – Self mod. DR output

1st call selfmod()

```
0x0000000140001e5b  push   rbx
0x0000000140001e5c  mov    rax, 0x00001234
0x0000000140001e63  mov    rbx, 0x00001234
0x0000000140001e6a  cmp    rax, rbx
0x0000000140001e6d  jz     0x0000000140001e78
0x0000000140001e78  mov    rax, 0x00000001
0x0000000140001e7f  inc    rax
0x0000000140001e82  call   0x0000000140001e53
0x0000000140001e87  nop
0x0000000140001e88  lea    rsi, <rel> [0x0000000140001eb8]
0x0000000140001e8f  mov    byte ptr [rsi], 0x48
0x0000000140001e92  mov    byte ptr [rsi+0x01], 0xc7
0x0000000140001e96  mov    byte ptr [rsi+0x02], 0xc0
0x0000000140001e9a  mov    byte ptr [rsi+0x03], 0x03
0x0000000140001e9e  mov    byte ptr [rsi+0x04], 0x00
0x0000000140001ea2  mov    byte ptr [rsi+0x05], 0x00
0x0000000140001ea6  mov    byte ptr [rsi+0x06], 0x00
0x0000000140001eaa  nop
0x0000000140001eab  nop
0x0000000140001eac  inc    rax
0x0000000140001eaf  dec    rax
0x0000000140001eb2  inc    rax
0x0000000140001eb5  dec    rax
0x0000000140001eb8  mov    rax, 0x00000003
0x0000000140001ebf  test   rax, rax
0x0000000140001ec2  jz     0x0000000140001ec6
0x0000000140001ec4  jnz    0x0000000140001ec9
0x0000000140001ec9  jmp    0x0000000140001ed7
0x0000000140001ed7  inc    rax
0x0000000140001eda  dec    rax
0x0000000140001edd  push   rsi
0x0000000140001ede  lea    rsi, <rel> [0x0000000140001e5c]
0x0000000140001ee5  mov    byte ptr [rsi], 0x48
0x0000000140001ee8  mov    byte ptr [rsi+0x01], 0xc7
0x0000000140001eec  mov    byte ptr [rsi+0x02], 0xc0
0x0000000140001ef0  mov    byte ptr [rsi+0x03], 0x01
0x0000000140001ef4  mov    byte ptr [rsi+0x04], 0x00
```

2nd call selfmod()

```
0x0000000140001e5b  push   rbx
0x0000000140001e5c  mov    rax, 0x00000001
0x0000000140001e63  mov    rbx, 0x00001234
0x0000000140001e6a  cmp    rax, rbx
0x0000000140001e6d  jz     0x0000000140001e78
0x0000000140001e6f  mov    rax, 0x00000000
0x0000000140001e76  jmp    0x0000000140001edd
0x0000000140001edd  push   rsi
0x0000000140001ede  lea    rsi, <rel> [0x0000000140001e5c]
0x0000000140001ee5  mov    byte ptr [rsi], 0x48
0x0000000140001ee8  mov    byte ptr [rsi+0x01], 0xc7
0x0000000140001eec  mov    byte ptr [rsi+0x02], 0xc0
0x0000000140001ef0  mov    byte ptr [rsi+0x03], 0x01
0x0000000140001ef4  mov    byte ptr [rsi+0x04], 0x00
0x0000000140001ef8  mov    byte ptr [rsi+0x05], 0x00
0x0000000140001efc  mov    byte ptr [rsi+0x06], 0x00
0x0000000140001f00  pop    rsi
0x0000000140001f01  pop    rbx
0x0000000140001f02  ret
```

Pre-execution
Self modification
48 C7 C0 03 00 00 00 = 'mov rax, 3'

```
printf("[ANTI-X] Selfmod return value: 0x%x (should be 0x3)\n", selfmodify());
printf("[ANTI-X] Selfmod return value: 0x%x (should be 0x0)\n", selfmodify());
```

```
[ANTI-X] Selfmod return value: 0x3 (should be 0x3)
[ANTI-X] Selfmod return value: 0x0 (should be 0x0)
```

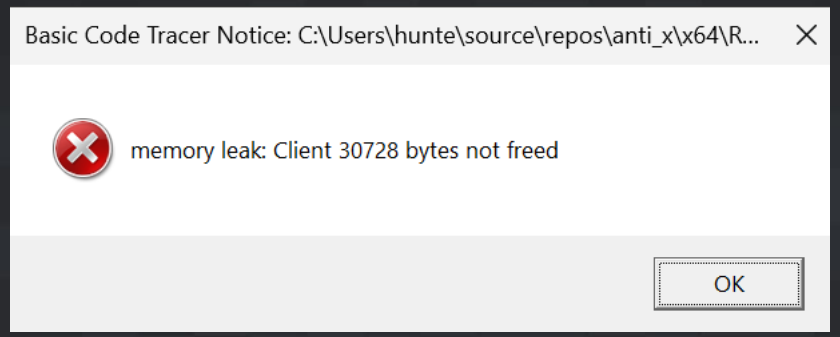
Post-execution
Self modification
48 C7 C0 01 00 00 00 = 'mov rax, 1'

Troubleshooting

Debugging the client

Debugging the client - Logging

Finding Memory leaks in your client



drrun.exe -debug -loglevel 2 -c client.dll – sample.exe

See official docs for log level description. Logs can be found at <DR_INSTALL_DIR>\logs

Hint: You do not need to compile the debug version of DynamoRIO from source.
The Release version from the Download website includes debugging features !

https://dynamorio.org/page_logging.html

Debugging DynamoRio (Client)

Last exit… WinDbg

Docs:
https://dynamorio.org/page_debugging.html#autotoc_md153

```
0:003> ~*kb

#  0  Id: a54.1520 Suspend: 1 Teb: 00000019`58b18000 Unfrozen
 # RetAddr           : Args to Child                                                     : Call Site
00 00007ff6`f44318e6     : 00000000`00000000 00007fff`0fe90057 00000000`00000000 00000195`43385700 :
drmgr!drmgr_bb_event_do_instrum_phases+0x42 [D:\a\dynamorio\dynamorio\ext\drmgr\drmgr.c @ 931]
01 00000000`710557c4     : 00000000`00000001 00000195`433a4ac0 00000000`00000000 00000000`00000000 : drmgr!drmgr_bb_event+0x276
[D:\a\dynamorio\dynamorio\ext\drmgr\drmgr.c @ 1198]
02 00000000`71093d1a     : 00000195`433a4e40 00000000`00000000 00000195`43385780 00000000`00000000 :
dynamorio!instrument_basic_block+0x124 [D:\a\dynamorio\dynamorio\core\lib\instrument.c @ 1769]
03 00000000`71093220     : 00000195`433a4e40 00000195`43385700 00000000`710a5700 00000000`43385700 : dynamorio!client_process_bb+0xca
[D:\a\dynamorio\dynamorio\core\arch\interp.c @ 2766]
04 00000000`710915dc     : 00000195`43385780 00000000`00000000 00007fff`0fe90000 00000195`00000000 : dynamorio!build_bb_ilist+0x17a0
[D:\a\dynamorio\dynamorio\core\arch\interp.c @ 4140]
05 00000000`7101e3fe     : 00000000`00000002 00000000`710047e0 00000000`00000000 00000195`43385780 :
dynamorio!build_basic_block_fragment+0x1cc [D:\a\dynamorio\dynamorio\core\arch\interp.c @ 5132]
06 00007ff6`d43be71b     : 00000000`00000287 00000019`58cfed69 00000000`00000000 00000000`00000000 : dynamorio!d_r_dispatch+0x3ce
[D:\a\dynamorio\dynamorio\core\dispatch.c @ 213]
07 00000000`00000287     : 00000019`58cfed69 00000000`00000000 00000000`00000000 00000000`00000000 : 0x00007ff6`d43be71b
08 00000019`58cfed69     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : 0x287
09 00000000`00000000     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : 0x00000019`58cfed69

   1  Id: a54.1560 Suspend: 1 Teb: 00000019`58b1a000 Unfrozen
 # RetAddr           : Args to Child                                                     : Call Site
00 00007fff`8fc9d31e     : 00000000`00000000 00000000`00000000 00007fff`8fcb0810 00000195`4309bba0 : ntdll!NtWaitForWorkViaWorkerFactory+0x14
01 00007fff`8f88e8d7     : 00000000`00000000 00000000`00000000 00000000`00000000 00000000`00000000 : ntdll!TppWorkerThread+0x37e
…
```

Building DynamoRio from Source

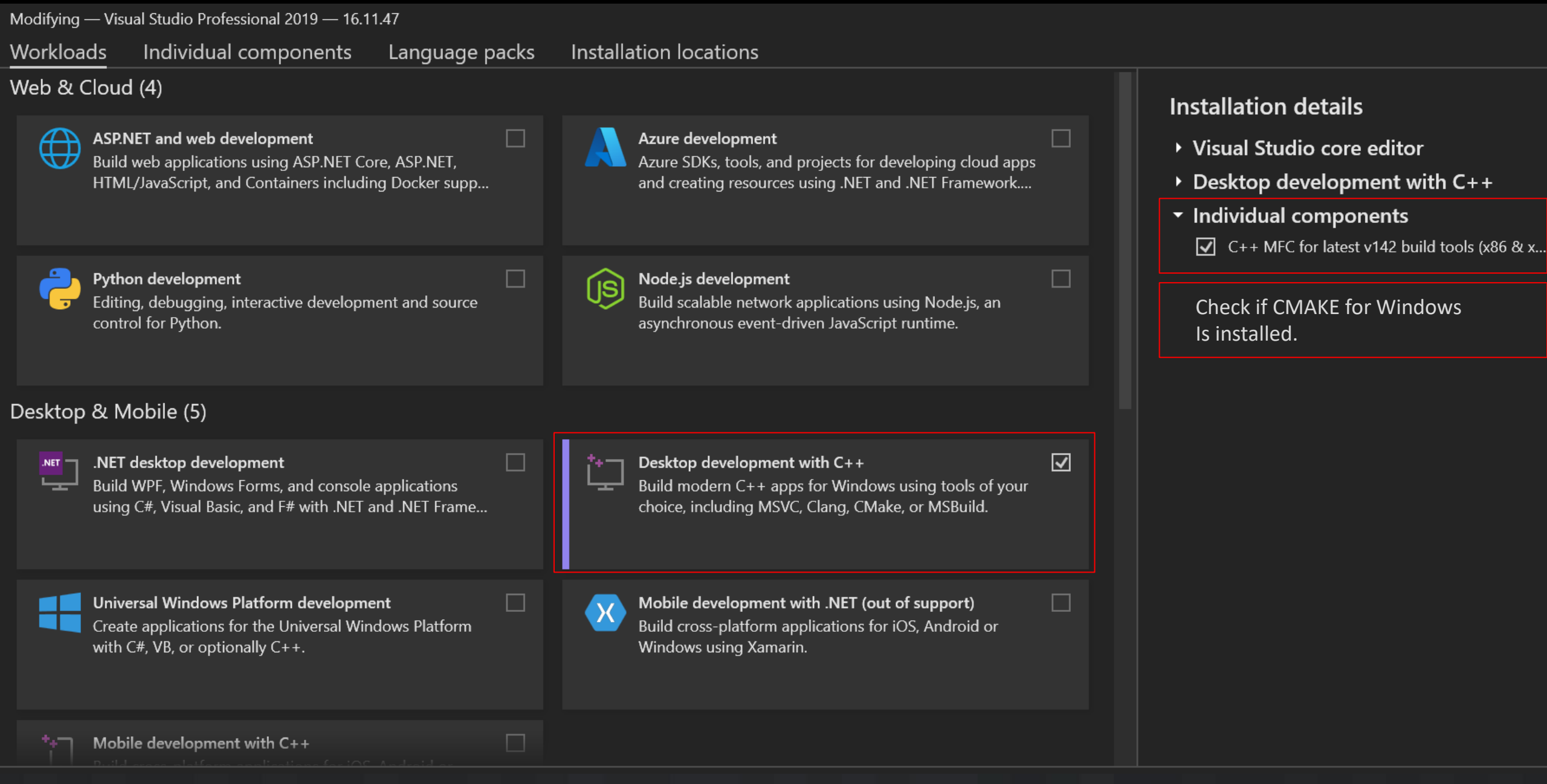
Windows Build Requirements

What you need to install first

- Visual Studio 2019. Other versions are not officially supported as our automated tests use VS 2019.
- CMake. 3.7+ is required. When prompted, we recommend adding it to your PATH.
- Git. Any flavor should do, including Git on Windows or Cygwin git.
- Perl. We recommend either Strawberry Perl or Cygwin perl.
- (optional) Cygwin. Only needed for building documentation. Select the doxygen package if you do install it. (You can also select Cygwin's perl and git as alternatives to the links above.)

Official docs:
https://dynamorio.org/page_building.html

VisualStudio Install



Compiling DynamoRio

Windows 11 24H2 [Version 10.0.26100.3915]

1) Install Visual Studio 2019 (only officially supported version), CMAKE [and doxygen]*

2a) Open Developer Command Prompt for VS 2019
2b) git clone --recurse-submodules https://github.com/DynamoRIO/dynamorio.git C:/tools/dynamorio-git

3) cd C:/tools/dynamorio-git
4) mkdir build && cd build

5a) Release : cmake -G"Visual Studio 16 2019" -A x64 ..
5b) Debug : cmake -G"Visual Studio 16 2019" -A x64 -DDEBUG:BOOL=ON ..

Other build options: next slide

Don't miss the two dots ☺

6a) Release : cmake --build . --config RelWithDebInfo
6b) Debug : cmake --build . --config Debug

```
c:\tools\dynamorio-git>cat CMakeLists.txt | grep CMAKE_CONFIGURATION_TYPES
   set(CMAKE_CONFIGURATION_TYPES "Debug" CACHE STRING "" FORCE)
   set(CMAKE_CONFIGURATION_TYPES "RelWithDebInfo" CACHE STRING "" FORCE)
```

7) cd C:/tools/dynamorio-git/exports/bin64

By default, compiled binaries are installed to <DYNRIO-GIT-DIR>/exports

Compiling DynamoRio

CMAKE config switches from DR docs

BUILD_CORE = whether to build the core
BUILD_DOCS = whether to build the documentation
BUILD_DRSTATS = whether to build the DRstats viewer (Windows-only).*
BUILD_TOOLS = whether to build the tools (primarily Windows)
BUILD_SAMPLES = whether to build the client samples
BUILD_TESTS* = whether to build the tests
INSTALL_PREFIX = where to install the results after building
NTDLL_LIBPATH = where ntdll.lib is located (Windows-only)
DEBUG* = whether to enable asserts and logging
INTERNAL = for DynamoRIO developer use
DISABLE_WARNINGS = useful if your version of gcc produces warnings we have not seen
BUILD_PACKAGE* = Build directory structure like official release

***off by default**

Build example – Full install

Simplified batch file – Installs executables to <DR-INSTALL>/install

```
REM --- x64 Release ---
cd C:\tools\dynamorio-git
mkdir build64rel && cd build64rel
cmake -G "Visual Studio 16 2019" -A x64 -DBUILD_PACKAGE=ON -DCMAKE_INSTALL_PREFIX=../install ..
cmake --build . --config RelWithDebInfo --target install

REM --- x64 Debug ---
cd C:\tools\dynamorio-git
mkdir build64dbg && cd build64dbg
cmake -G "Visual Studio 16 2019" -A x64 -DBUILD_PACKAGE=ON -DCMAKE_INSTALL_PREFIX=../install -DDEBUG:BOOL=ON ..
cmake --build . --config Debug --target install

REM --- x32 Release ---
cd C:\tools\dynamorio-git
mkdir build32rel && cd build32rel
cmake -G "Visual Studio 16 2019" -A win32 -DBUILD_PACKAGE=ON -DCMAKE_INSTALL_PREFIX=../install ..
cmake --build . --config RelWithDebInfo --target install

REM --- x32 Debug ---
cd C:\tools\dynamorio-git
mkdir build32dbg && cd build32dbg
cmake -G "Visual Studio 16 2019" -A win32 -DBUILD_PACKAGE=ON -DCMAKE_INSTALL_PREFIX=../install -DDEBUG:BOOL=ON ..
cmake --build . --config Debug --target install
```

Summary

Many Anti-X techniques work out of the box

- DR is not the silver bullet, but it is easy, fast and transparent for most operations
- Multi-threating and child process compatible
- Anti-X
  - Shellcode
  - SSL/TLS intercepting
  - Code Validation Checks
  - Large Loops
  - Simple Anti-Debug checks
  - Complex Anti-Debug checks
  - Exception Handling
  - Runtime checks
  - Self modifying code…

TALOSINTELLIGENCE.COM

blog.talosintelligence.com

@hunterbr72

Thank you!

TALOSINTELLIGENCE.COM

blog.talosintelligence.com

@talossecurity

TBD:

https://drmemory.org/page_drstrace.html

https://drmemory.org/page_symquery.html