# Frida快速入门

## 课前准备

1. 学员基础：
   1. 了解汇编语言
   2. 了解Python
   3. 熟悉TypeScript
   4. 了解JavaScript
   5. C/C++(高级部分)
2. 演示环境：
   1. VMware Workstation Pro
   2. Windows 10
   3. PyCharm
   4. VSCode
   5. Visual Studio 2022
   6. OD
   7. CE
   8. IDA
3. 演示目标：winmine.exe (windows xp系统附带的扫雷游戏)
   1. 修改时间：2001‎年‎8‎月‎31‎日，‏‎16:04:32
   2. **软件版本：5.1.2600.0**
   3. **软件大小：117 KB (119,808 字节)**
4. 课程源码：https://github.com/zmrbak/Frida
5. 参考资料：https://frida.re/
6. 课程类别：演示、实操教学

## Frida安装

https://frida.re/

<https://www.python.org/>

<https://www.jetbrains.com/>

<https://pc.weixin.qq.com/>

## Frida中的Python脚本与JavaScript脚本

## Frida中的TypeScript脚本与JavaScript脚本（1）

<https://frida.re/>

<https://nodejs.org/en/>

<https://github.com/oleavr/frida-agent-example>

## Frida中的TypeScript脚本与JavaScript脚本（2）

## 棋盘数据定位与读取

CE

C:\Program Files\Cheat Engine 7.4\languages

<https://cheatengine.org/downloads.php>

宽度

winmine.exe+5334 - 16 - push ss

高度

winmine.exe+5338 - 15 00000000 - adc eax,00000000 { 0 }

雷数

winmine.exe+5330 - 2A 00 - sub al,[eax]



棋盘头

winmine.exe+5340 - 10 10 - adc [eax],dl

<?xml version="1.0" encoding="utf-8"?>

<CheatTable>

<CheatEntries>

<CheatEntry>

<ID>4</ID>

<Description>"高度"</Description>

<LastState Value="10" RealAddress="01005338"/>

<ShowAsSigned>0</ShowAsSigned>

<VariableType>4 Bytes</VariableType>

<Address>winmine.exe+5338</Address>

</CheatEntry>

<CheatEntry>

<ID>3</ID>

<Description>"宽度"</Description>

<LastState Value="20" RealAddress="01005334"/>

<ShowAsSigned>0</ShowAsSigned>

<VariableType>4 Bytes</VariableType>

<Address>winmine.exe+5334</Address>

</CheatEntry>

<CheatEntry>

<ID>5</ID>

<Description>"雷数"</Description>

<LastState Value="30" RealAddress="01005330"/>

<ShowAsSigned>0</ShowAsSigned>

<VariableType>4 Bytes</VariableType>

<Address>winmine.exe+5330</Address>

</CheatEntry>

<CheatEntry>

<ID>2</ID>

<Description>"棋盘头"</Description>

<LastState Value="10101010" RealAddress="01005340"/>

<ShowAsHex>1</ShowAsHex>

<ShowAsSigned>0</ShowAsSigned>

<VariableType>4 Bytes</VariableType>

<Address>winmine.exe+5340</Address>

</CheatEntry>

</CheatEntries>

</CheatTable>

## Frida读取棋盘数据

## 分析和修改棋盘数据

====================== 2022-06-12T04:23:14.717Z ==========================

Frida.version 15.1.24

棋盘高度: 9

棋盘宽度: 9

地雷数量: 10

棋盘头: 0x1005340

10 10 10 10 10 10 10 10 10 10 10

10 0F 0F 0F 0F 8F 0F 0F 0F 0F 10

10 0F 0F 0F 8F 0F 0F 0F 0F 0F 10

10 0F 0F 0F 0F 0F 0F 8F 0F 8F 10

10 8F 0F 0F 0F 0F 0F 0F 0F 0F 10

10 0F 0F 0F 8F 8F 0F 0F 0F 0F 10

10 0F 0F 0F 0F 0F 8F 0F 0F 8F 10

10 0F 0F 0F 0F 0F 0F 0F 0F 0F 10

10 0F 0F 0F 0F 0F 8F 0F 0F 0F 10

10 0F 0F 0F 0F 0F 0F 0F 0F 0F 10

10 10 10 10 10 10 10 10 10 10 10

10：棋盘边框

**游戏开始之前：**

**8F：地雷**

**0F：不是雷（空的）**

**游戏开始之后：**

**8E：做了标记（对的）**

**0E：标错的地方（错的）**

**游戏结束**

**0F：不是雷（空的）**

**CC：爆掉的地雷**

**8A：未爆的地雷（游戏结束）**

**8E：做了标记**

**41:1**

**42:2**

**。。。**

**48:8**

## 让系统重新绘制指定窗口

**SpyXX.exe**

**C:\Program Files\Microsoft Visual Studio\2022\Preview\Common7\Tools**

1. **获取窗口句柄**

**FindWindow(不好用，最顶层符合条件的窗口)**

<?xml version="1.0" encoding="utf-8"?>

<CheatTable>

<CheatEntries>

<CheatEntry>

<ID>0</ID>

<Description>"窗口句柄"</Description>

<LastState Value="001904C0" RealAddress="01005B24"/>

<ShowAsHex>1</ShowAsHex>

<VariableType>4 Bytes</VariableType>

**<Address>winmine.exe+5B24</Address>**

</CheatEntry>

</CheatEntries>

</CheatTable>

1. 获取窗口客户区域

GetClientRect

3、将这一部分区域重新绘制

InvalidateRect

private GetClientRect!: NativePointer | null;

private InvalidateRect!: NativePointer | null;

this.GetClientRect = Module.findExportByName("User32.dll", "GetClientRect");

this.InvalidateRect = Module.findExportByName("User32.dll", "InvalidateRect");

// BOOL GetClientRect(

// [in] HWND hWnd,

// [out] LPRECT lpRect

// );

const lpRect = Memory.alloc(4 \* 4);

let GetClientRect = new NativeFunction(this.GetClientRect!, "bool", ["pointer", "pointer"]);

let result = GetClientRect(this.hWnd, lpRect);

console.log("result", result);

// BOOL InvalidateRect(

// [in] HWND hWnd,

// [in] const RECT \* lpRect,

// [in] BOOL bErase

// );

let InvalidateRect = new NativeFunction(this.InvalidateRect!, "bool", ["pointer", "pointer", 'bool']);

result = InvalidateRect(this.hWnd, lpRect, 1);

console.log("result", result);

## Frida中TypeScript导入模块的问题

$ git clone git://github.com/oleavr/frida-agent-example.git

$ cd frida-agent-example/

$ npm install

$ frida -U -f com.example.android --no-pause -l \_agent.js

$ npm run watch

import { WinApi } from "./win\_api";

export class WinApi {}

## TypeScript模块的封装

## 使用Frida将目标窗口切换到前台（1）

board\_foreground() {

let hForeWnd = WinApi.GetForegroundWindow();

let dwCurID = WinApi.GetCurrentThreadId();

let dwForeID = WinApi.GetWindowThreadProcessId(hForeWnd, ptr(0));

WinApi.AttachThreadInput(dwCurID, dwForeID, 1);

const SW\_RESTORE = 9;

WinApi.ShowWindow(this.hWnd, SW\_RESTORE);

WinApi.SetForegroundWindow(this.hWnd);

const HWND\_TOPMOST = -1;

const HWND\_NOTOPMOST = -2;

const SWP\_NOSIZE = 0x0001;

const SWP\_NOMOVE = 0x0002;

WinApi.SetWindowPos(this.hWnd, HWND\_TOPMOST, 0, 0, 0, 0, SWP\_NOSIZE | SWP\_NOMOVE);

WinApi.SetWindowPos(this.hWnd, HWND\_NOTOPMOST, 0, 0, 0, 0, SWP\_NOSIZE | SWP\_NOMOVE);

WinApi.AttachThreadInput(dwCurID, dwForeID, 0);

}

## 使用Frida将目标窗口切换到前台（2）

## TypeScript模块优化

private static func\_GetWindowThreadProcessId: AnyFunction;

static GetWindowThreadProcessId(hWnd: NativePointerValue, lpdwProcessId: NativePointerValue): number {

if (this.func\_GetWindowThreadProcessId == undefined) {

let address = Module.findExportByName("User32.dll", "GetWindowThreadProcessId");

this.func\_GetWindowThreadProcessId = new NativeFunction(address!, "int", ["pointer", "pointer"]);

}

return this.func\_GetWindowThreadProcessId(hWnd, lpdwProcessId);

}

private static func\_AAA: AnyFunction;

static AAA(hWnd: NativePointerValue, lpdwProcessId: NativePointerValue): number {

if (this.func\_AAA == undefined) {

let address = Module.findExportByName("User32.dll", "AAA");

this.func\_AAA = new NativeFunction(address!, "int", ["pointer", "pointer"]);

}

return this.func\_AAA(hWnd, lpdwProcessId);

}

## 获取软件窗口位置，设置鼠标指针位置

board\_location() {

let lpOrgRect = Memory.alloc(4 \* 4);

WinApi.GetCursorPos(lpOrgRect);

// typedef struct tagRECT {

// LONG left;

// LONG top;

// LONG right;

// LONG bottom;

// } RECT, \*PRECT, \*NPRECT, \*LPRECT;

let lpRect = Memory.alloc(4 \* 4);

WinApi.GetWindowRect(this.hWnd, lpRect);

console.log("left", lpRect.readU32());

console.log("top", lpRect.add(4).readU32());

console.log("right", lpRect.add(8).readU32());

console.log("bottom", lpRect.add(12).readU32());

WinApi.SetCursorPos(lpRect.readU32(), lpRect.add(4).readU32());

**不可用Thread.sleep(),否则会导致应用程序故障**

WinApi.Sleep(2000);

WinApi.SetCursorPos(lpOrgRect.readU32(), lpOrgRect.add(4).readU32());

}

## 设置鼠标位置，自动点击鼠标

void \_\_stdcall mouse\_event(DWORD dwFlags, DWORD dx, DWORD dy, DWORD dwData, ULONG\_PTR dwExtraInfo)

{

int v5[7]; // [esp+0h] [ebp-1Ch] BYREF

v5[0] = 0;

v5[1] = dx;

v5[2] = dy;

v5[3] = dwData;

v5[4] = dwFlags;

v5[5] = 0;

v5[6] = dwExtraInfo;

**NtUserSendInput**(1, v5, 28);

}

// attributes: thunk

UINT \_\_stdcall **NtUserSendInput**(UINT cInputs, LPINPUT pInputs, int cbSize)

{

return \_\_imp\_\_NtUserSendInput@12(cInputs, pInputs, cbSize);

}

const MOUSEEVENTF\_LEFTDOWN = 0x0002;

const MOUSEEVENTF\_LEFTUP = 0x0004;

const MOUSEEVENTF\_RIGHTDOWN = 0x0008;

const MOUSEEVENTF\_RIGHTUP = 0x0010;

WinApi.MouseEvent(MOUSEEVENTF\_LEFTDOWN, 0, 0, 0, WinApi.GetMessageExtraInfo());

WinApi.MouseEvent(MOUSEEVENTF\_LEFTUP, 0, 0, 0, WinApi.GetMessageExtraInfo());

// WinApi.MouseEvent(MOUSEEVENTF\_RIGHTDOWN, 0, 0, 0, WinApi.GetMessageExtraInfo());

// WinApi.MouseEvent(MOUSEEVENTF\_RIGHTUP, 0, 0, 0, WinApi.GetMessageExtraInfo());

## 用鼠标自动标记棋盘上的雷区

board\_click() {

//记录鼠标位置

let lpOrgRect = Memory.alloc(4 \* 4);

WinApi.GetCursorPos(lpOrgRect);

//加载棋盘数据

this.load\_board\_info();

//获取棋盘位置

let lpRect = Memory.alloc(4 \* 4);

WinApi.GetWindowRect(this.hWnd, lpRect);

this.start\_x = lpRect.readU32() + 7;

this.start\_y = lpRect.add(4).readU32() + 92;

//遍历棋盘，按行遍历

for (let i = 1; i < this.height + 2; i++) {

//按列遍历

for (let j = 1; j < this.width + 2; j++) {

let byte\_data = this.head.add(j + 0x20 \* i).readU8();

//标记地雷

if (byte\_data == 0x8F) {

this.mouse\_click(j, i, false);

continue;

}

//点击无雷区

if (byte\_data == 0x0F) {

this.mouse\_click(j, i);

continue;

}

}

}

//鼠标归位

WinApi.SetCursorPos(lpOrgRect.readU32(), lpOrgRect.add(4).readU32());

}

## Frida命名空间的使用

let version = Frida.version;

console.log(version);

let heapSize = Frida.heapSize;

console.log(heapSize);

## Process命名空间的使用

Process: 进程

进程（Process）是计算机中的程序关于某数据集合上的一次运行活动，是系统进行资源分配和调度的基本单位，是操作系统结构的基础。在早期面向进程设计的计算机结构中，进程是程序的基本执行实体；在当代面向线程设计的计算机结构中，进程是线程的容器。程序是指令、数据及其组织形式的描述，进程是程序的实体

show\_process() {

console.log("Process.id:\t\t", Process.id);

console.log("Process.arch:\t\t", Process.arch);

console.log("Process.platform:\t", Process.platform);

console.log("Process.pageSize:\t", Process.pageSize);

console.log("Process.pointerSize:\t", Process.pointerSize);

console.log("Process.codeSigningPolicy:\t", Process.codeSigningPolicy);

console.log("Process.isDebuggerAttached():\t", Process.isDebuggerAttached());

console.log("Process.getCurrentThreadId():\t", Process.getCurrentThreadId());

console.log("Process.getCurrentThreadId():\t", Process.getCurrentThreadId());

// let threads = Process.enumerateThreads();

// for (const iterator of threads) {

// console.log(JSON.stringify(iterator));

// }

// let modules = Process.enumerateModules();

// for (const iterator of modules) {

// console.log(JSON.stringify(iterator));

// }

// let ranges = Process.enumerateRanges("rwx");

// for (const iterator of ranges) {

// console.log(JSON.stringify(iterator));

// }

let mallocRanges = Process.enumerateMallocRanges();

for (const iterator of mallocRanges) {

console.log(JSON.stringify(iterator));

}

}

## Process.setExceptionHandler的使用

show\_process() {

Process.setExceptionHandler((exception) => {

console.log(JSON.stringify(exception, null, 4));

return false;

});

}

## Module类的使用

demo() {

// let module = Process.getModuleByName("winmine.exe");

// let module = Process.getModuleByName("user32.dll");

let module = Process.getModuleByName("Kernel32.dll");

// console.log("module", JSON.stringify(module, null, 4));

// console.log("Imports:");

// for (const iterator of module.enumerateImports()) {

// console.log(JSON.stringify(iterator));

// }

// console.log("Exports:");

// for (const iterator of module.enumerateExports()) {

// console.log(JSON.stringify(iterator));

// }

// console.log("Symbols:");

// for (const iterator of module.enumerateSymbols()) {

// console.log(JSON.stringify(iterator));

// }

//enumerateRanges

// console.log("Ranges:");

// for (const iterator of module.enumerateRanges("r--")) {

// console.log(JSON.stringify(iterator));

// }

//{"type":"function","name":"lstrlenW","address":"0x7630e0b0"}

// let p = module.findExportByName("lstrlenW");

// console.log(p);

// let p1 = Module.load("C:\\DBGHELP.DLL");

// console.log(JSON.stringify(p1));

// console.log("Exports:");

// for (const iterator of p1.enumerateExports()) {

// console.log(JSON.stringify(iterator));

// }

// console.log("Imports:");

// for (const iterator of p1.enumerateImports()) {

// console.log(JSON.stringify(iterator));

// }

}

## NativePointer类的使用（1）

指针：

数据

代码

数字：8位，16位，32位，64位。。。，

指针

字符

ASCII

CString

UTF8

UTF16

UTF32

## NativePointer类的使用（2）

demo() {

let str1 = "This is a string! 你好！";

console.log(str1);

let m1 = Memory.alloc(4);

let m2 = Memory.allocAnsiString(str1);

let m3 = Memory.allocUtf16String(str1);

let m4 = Memory.allocUtf8String(str1);

console.log(m2);

// console.log(m2.readAnsiString());

// console.log(m2.readCString());

// console.log(m2.readUtf8String());

// console.log(m2.readUtf16String());

let ab = m2.readByteArray(str1.length \* 2 + 2);

// console.log(ab?.byteLength);

let i8 = new Int8Array(ab!);

// let i8 = new Int32Array(ab!);

// let i8 = new String(ab!);

// for (const iterator of i8) {

// console.log(iterator);

// }

// console.log("b1", i8.map(x => String.fromCharCode(x)).join(""));

let b1 = Array.prototype.slice.call(new Int8Array(ab!));

console.log("b1", b1.map(x => String.fromCharCode(x)).join(""));

// console.log(ab);

// console.log(m2.readPointer());

// console.log(m2.readS8().toString(16));

// console.log(m2.readFloat());

// console.log(m2.readDouble());

// m1.writeU8(0xF8);

// console.log(m1.readU8());

// console.log(m1.readS8());

// let p1 = ptr(1);

// console.log(p1.isNull());

// console.log(p1.add(100).toInt32());

// let p2 = new NativePointer("0x12345678");

// let p3 = ptr(1);

// let p4 = ptr(0x11223344);

// console.log(p3 == p4);

// console.log(p3.equals(p4));

// console.log(p3.compare(p4));

// console.log("3".localeCompare("3"));

// console.log(p4);

// console.log(p4.toString());

// console.log(p4.toJSON());

// console.log(p4.toMatchPattern());

}

## Memory命名空间的使用（1）

demo() {

// console.log(JSON.stringify(Memory));

let module = Process.getModuleByName("winmine.exe");

let p = ptr(0x00210604);

let pattern = p.toMatchPattern();

console.log("pattern", pattern);

// Memory.scan(module.base, module.size, pattern, {

// Memory.scan(module.base, module.size, "04 ?? ?1 ?0", {

// onMatch: (address, size) => {

// console.log("onMatch", size, address, address.sub(module.base));

// },

// onError: (reason) => {

// console.log(reason);

// },

// onComplete: () => {

// console.log("Scan Complete!");

// }

// });

// let matches = Memory.scanSync(module.base, module.size, pattern);

// let matches = Memory.scanSync(module.base, module.size, "04 ?? ?1 ?0");

// for (const iterator of matches) {

// console.log(JSON.stringify(iterator));

// }

// let m1 = Memory.alloc(Process.pageSize);

// console.log("protect", JSON.stringify(Process.getRangeByAddress(m1)));

// Memory.protect(m1, Process.pageSize, "r-x");

// console.log("protect", JSON.stringify(Process.getRangeByAddress(m1)));

let lpText = Memory.allocUtf16String("This is a string！");

let lpCaption = Memory.allocUtf16String("Caption");

// WinApi.MessageBox(p, lpText, lpCaption, 0x00000001);

let m2 = Memory.alloc(Process.pageSize);

console.log("m2", m2);

let address = Module.getExportByName("User32.dll", "MessageBoxW");

Memory.patchCode(m2, Process.pageSize, (code) => {

// console.log("code", code);

let asm = new X86Writer(code);

asm.putPushU32(0x00000001);

asm.putPushU32(lpCaption.toUInt32());

asm.putPushU32(lpText.toUInt32());

// asm.putPushU32(p.toUInt32());

asm.putPushU32(0);

asm.putCallAddress(address);

asm.putRet();

asm.flush();

});

let func = new NativeFunction(m2, "void", []);

func();

}

## Memory命名空间的使用（2）

## Instruction类的反汇编操作

demo() {

let address = Module.getExportByName("User32.dll", "MessageBoxW");

let m2 = Memory.alloc(Process.pageSize);

let asm = new X86Writer(m2);

asm.putPushU32(0x00000001);

asm.putPushU32(2);

asm.putPushU32(3);

asm.putPushU32(4);

asm.putCallAddress(address);

asm.putRet();

asm.putPushReg("eax");

asm.flush();

// this.show\_asm(m2);

this.show\_asm(ptr(0x01003E21));

}

show\_asm(start: NativePointer, length: number = 10) {

for (let index = 0; index < length; index++) {

let inst = Instruction.parse(start);

// console.log(JSON.stringify(inst));

let byteArray = start.readByteArray(inst.size);

let byteCode = Array.prototype.slice.call(new Uint8Array(byteArray!));

let mCode = byteCode.map(x => x.toString(16).padStart(2, "0")).join(" ").toUpperCase();

console.log(inst.address.toString().toUpperCase().replace("0X", "0x"), mCode.padEnd(14, " "), "\t", inst.toString().toUpperCase().replace("0X", "0x"));

start = inst.next;

if (start.readU32() == 0) break;

}

}