Harder, Better, Faster, Stronger

Semi-Auto Vulnerability Research

Professional Vulnerability Research

- Finding bugs is not the problem
  - Fuzzing works
    - Microsoft found over 1800 bugs in Office 2010
      - http://blogs.technet.com/b/office2010/archive/2010/05/11/how-the-sdl-helped-improve-security-in-office-2010.aspx
    - 280 bugs found in Mozilla JavaScript using JSFunFuzz
      - https://bugzilla.mozilla.org/show_bug.cgi?id=jsfunfuzz
- Tooling is not the problem
  - Distributed fuzzing
  - Crash analyzers
- Lack of intelligent workflow is the problem

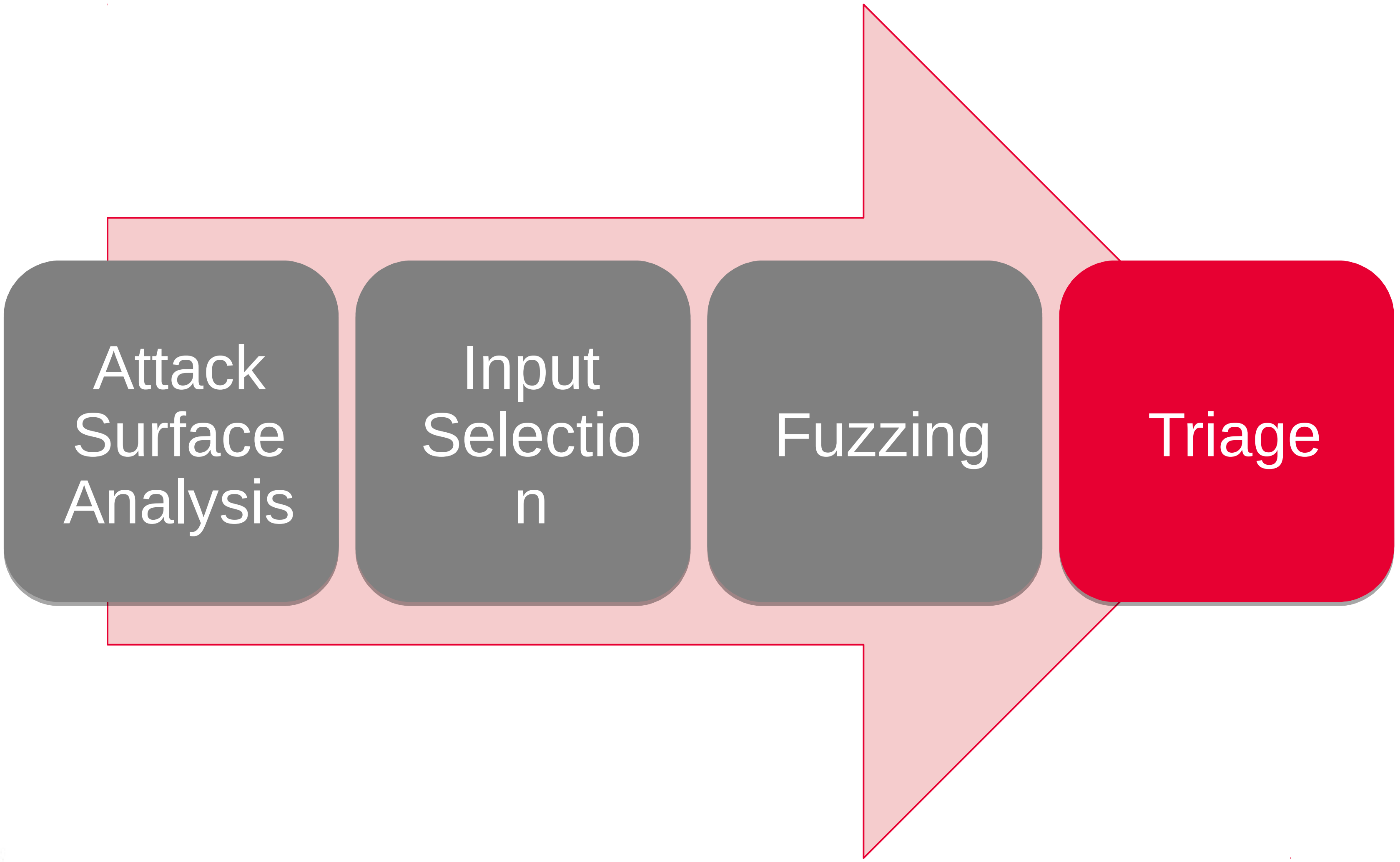
Main Goal

Develop an effective workflow and toolset for fuzzing and triaging vulnerabilities in a production environment

Ancillary Goals

- Primary
  - Determine cause and exploitability
  - Human time efficiency
- Secondary
  - CPU efficiency
  - Ease of use

Process Breakdown



Keys to Fuzzing Smartly

- Input selection
  - Most important factor in timely bug discovery
  - Time management
- Automation
  - SIMPLE Distributed fuzzing
  - Crash analysis
  - Bucketing
  - Confidence Rating

Keys to Smart Bug Triage

- Crash selection
  - Select for understanding
  - Crash database
  - Bug classes

- Program flow analysis
  - Code coverage
  - Input Mapping
  - Taint Analysis

Input Selection

- Attack Surface Analysis
  - Determine which areas of the code are reachable from external inputs

- Template code coverage
  - Determine what areas of code are exercised by different templates

- Rank templates based upon coverage of targeted code or overall attack surface

Fuzzing

- *The Miller Theorem*

The Miller Theorem

***C* ≡ code path coverage**

***T* ≡ Time spent Fuzzing**

***B* ≡ Bugs Discovered**

$C \equiv$ code path coverage

$T \equiv$ Time spent Fuzzing

$B \equiv$ Bugs Discovered

$$\partial C \cdot \partial T = \partial B$$

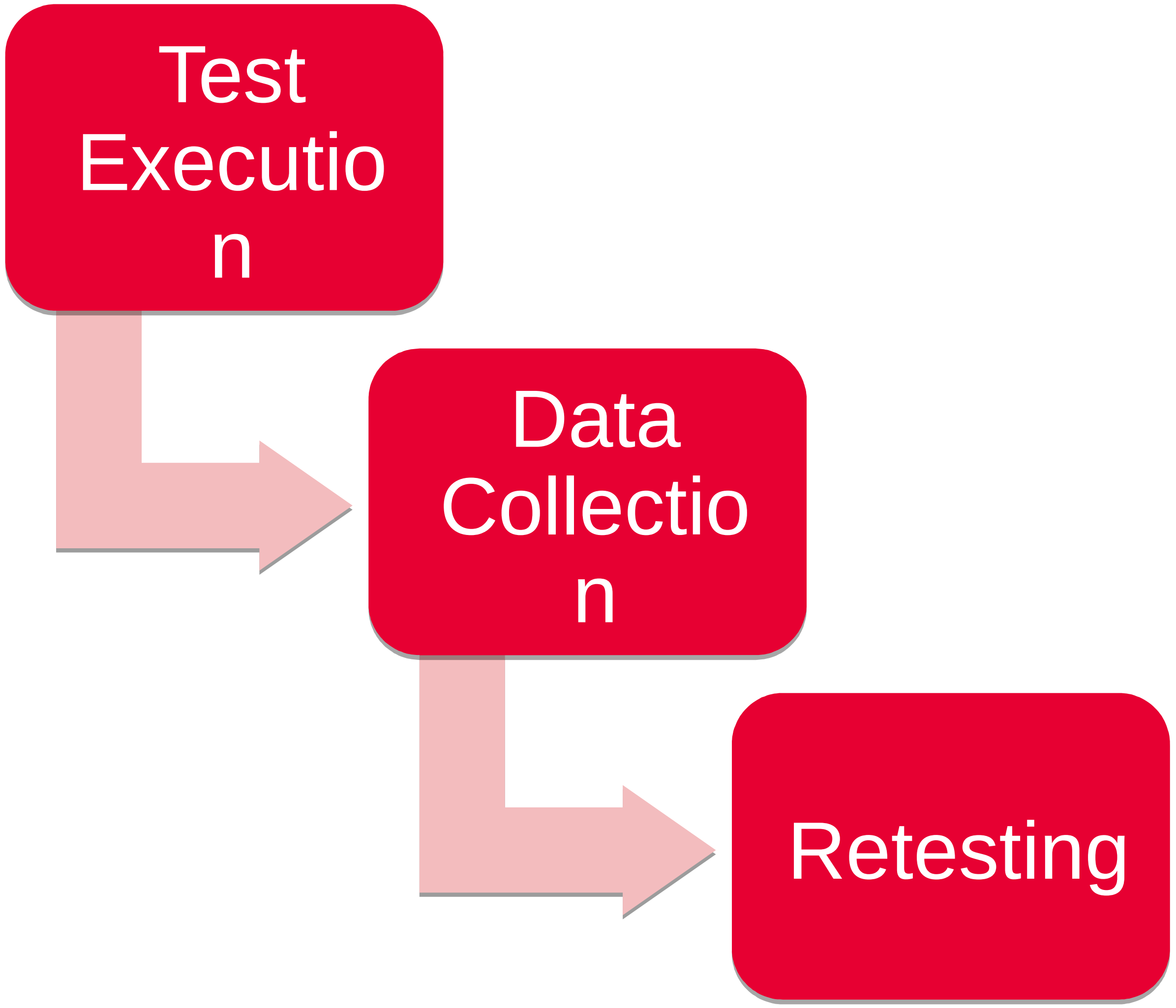
Fuzzing

- Obey the Miller Theorem
  - Create inputs to maximize coverage
  - Create the framework to maximize uptime

- Generation vs. Mutation
  - If you can, do both!
  - Mutation is cheaper, still works

- Do as little work as possible
  - Re-do as little work as possible

Fuzzing



Fuzzing – Test Execution

- Watch your tests well
  - Embedded custom debugger
  - Be able to gather needed data at crash time
  - Make use of debugging technologies
  - Be able to avoid invalid exceptions

- Distribute your tests
  - Centralized management
  - Make it easy to add nodes

Fuzzing – Data Storage

- Use a database!
  - Store lots of data over time
  - Easily searched
- What to store
  - Store what you need for crash selection
  - All crash information
  - Software versioning information
    - Binary diffs

Fuzzing - Retesting

- Maintaining a good database allows:
  - ▸ Automated retesting of modified code paths
  - ▸ Automated retesting of crashes in modified code paths

- Track bug life across software versions
  - ▸ A bug which lives through a nearby patch can have a long shelf-life
    - MS08-067 and MS06-040
    - ANI

Triage – Crash Selection

- Which crashes should receive priority?
- What properties make crashes more exploitable?
  - Knowledge! Familiarity!
- Crash database
  - Vulnerability properties
  - Searchable crash criteria

Triage – Crash Selection

- Exception Analysis
  - ▸ Determine level of access exception grants user

- Bug Class Identification
  - ▸ Difficulty of exploitability varies by bug class
  - ▸ Custom architecture problems
    - Custom memory allocators

Triage – Program Flow Analysis

- Abstract a program into flows
  - ► Code execution
  - ► Data dependency
- Code Coverage
  - ► Block hit trace for path to exception
  - ► Build a graph of program execution
  - ► Augment static program graphs

Triage – Program Flow Analysis

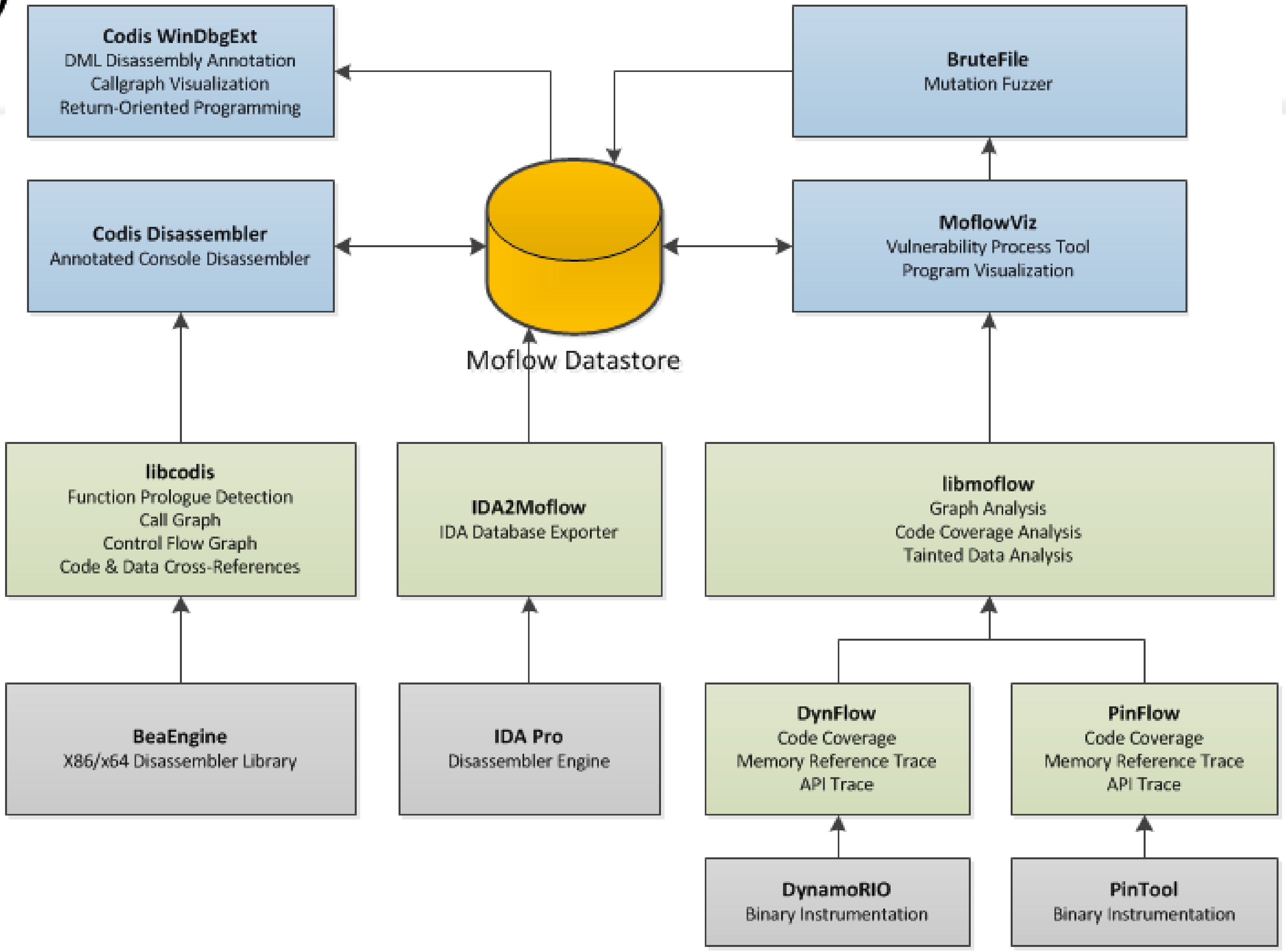
- Input Mapping
  - Trace APIs or System Calls that perform I/O
  - Mark data copied from external sources into memory

- Taint analysis
  - Follow input through the execution of the program
  - Determine where the bytes of the crash originated
  - Potential for exploit and signature generation

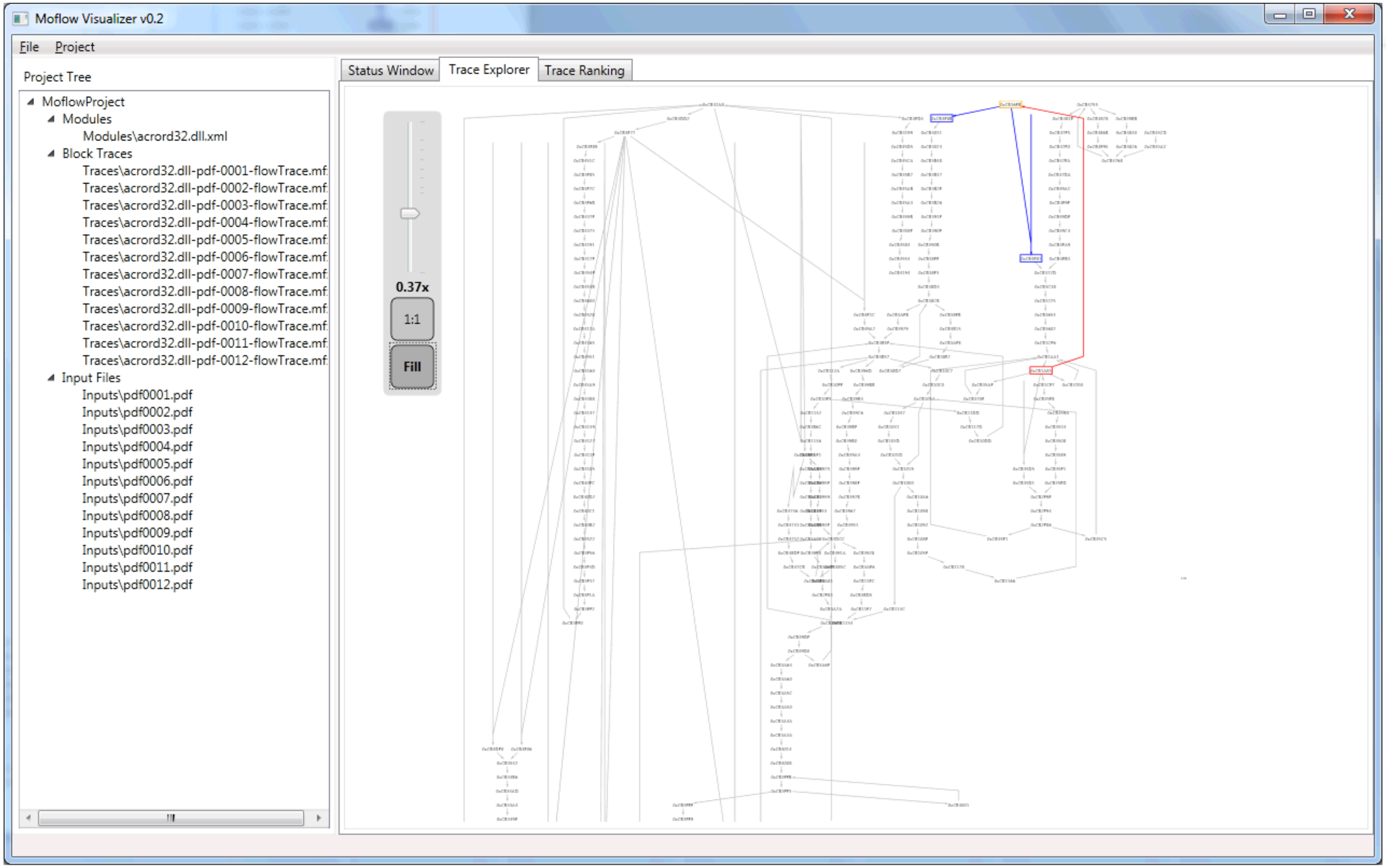
Triage – Program Flow Analysis

- Visualization
  - Provides a graphical representation of program structure and execution paths
  - Visualization allows overlaying multiple graphs and datasets using visual cues
  - Converting data to a visual problem allows rapid understanding of large datasets

Moflow



Codis WinDbgExt
DML Disassembly Annotation
Callgraph Visualization
Return-Oriented Programming
BruteFile
Mutation Fuzzer
Codis Disassembler
Annotated Console Disassembler
Moflow Datastore
MoflowViz
Vulnerability Process Tool
Program Visualization
libcodis
Function Prologue Detection
Call Graph
Control Flow Graph
Code & Data Cross-References
IDA2Moflow
IDA Database Exporter
libmoflow
Graph Analysis
Code Coverage Analysis
Tainted Data Analysis
BeaEngine
X86/x64 Disassembler Library
IDA Pro
Disassembler Engine
DynFlow
Code Coverage
Memory Reference Trace
API Trace
PinFlow
Code Coverage
Memory Reference Trace
API Trace
DynamoRIO
Binary Instrumentation
PinTool
Binary Instrumentation



Project Tree

- MoflowProject
  - Modules
    - Modules\acrord32.dll.xml
  - Block Traces
    - Traces\acrord32.dll-pdf-0001-flowTrace.mf
    - Traces\acrord32.dll-pdf-0002-flowTrace.mf
    - Traces\acrord32.dll-pdf-0003-flowTrace.mf
    - Traces\acrord32.dll-pdf-0004-flowTrace.mf
    - Traces\acrord32.dll-pdf-0005-flowTrace.mf
    - Traces\acrord32.dll-pdf-0006-flowTrace.mf
    - Traces\acrord32.dll-pdf-0007-flowTrace.mf
    - Traces\acrord32.dll-pdf-0008-flowTrace.mf
    - Traces\acrord32.dll-pdf-0009-flowTrace.mf
    - Traces\acrord32.dll-pdf-0010-flowTrace.mf
    - Traces\acrord32.dll-pdf-0011-flowTrace.mf
    - Traces\acrord32.dll-pdf-0012-flowTrace.mf
  - Input Files
    - Inputs\pdf0001.pdf
    - Inputs\pdf0002.pdf
    - Inputs\pdf0003.pdf
    - Inputs\pdf0004.pdf
    - Inputs\pdf0005.pdf
    - Inputs\pdf0006.pdf
    - Inputs\pdf0007.pdf
    - Inputs\pdf0008.pdf
    - Inputs\pdf0009.pdf
    - Inputs\pdf0010.pdf
    - Inputs\pdf0011.pdf
    - Inputs\pdf0012.pdf

Moflow: Input Selection

Input Selection - Requirements

- Attack Surface Analysis
  - Call graph analysis
- Template code coverage
  - Dynamic tracing
- Template ranking
  - Coverage graph analysis

Attack Surface Analysis

- Obtain call graph
  - IDA2Moflow.idc
  - LibCodis

- Define APIs that are data entry points

	Input Source	**I/O API**

Networ k	connect()
	accept()
		send()
		recv()

File	NtOpenFile()
	NtCreateFile()
	SYS_Open()
		NtReadFile()
		NtWriteFile()
		SYS_Read()
		SYS_Write()

Attack Surface Analysis

- Determine reachability graph from each API

```
δ-wavefront ← RootSet
closure ← ⟨⟩
While nonEmpty(δ-wavefront) Do
    wavefront ← oneStep(δ-wavefront)
    δ-wavefront ← wavefront − closure
    closure ← closure ∪ δ-wavefront
End While
Return closure
```

δ-wavefront Algorithm – Qadah et al.

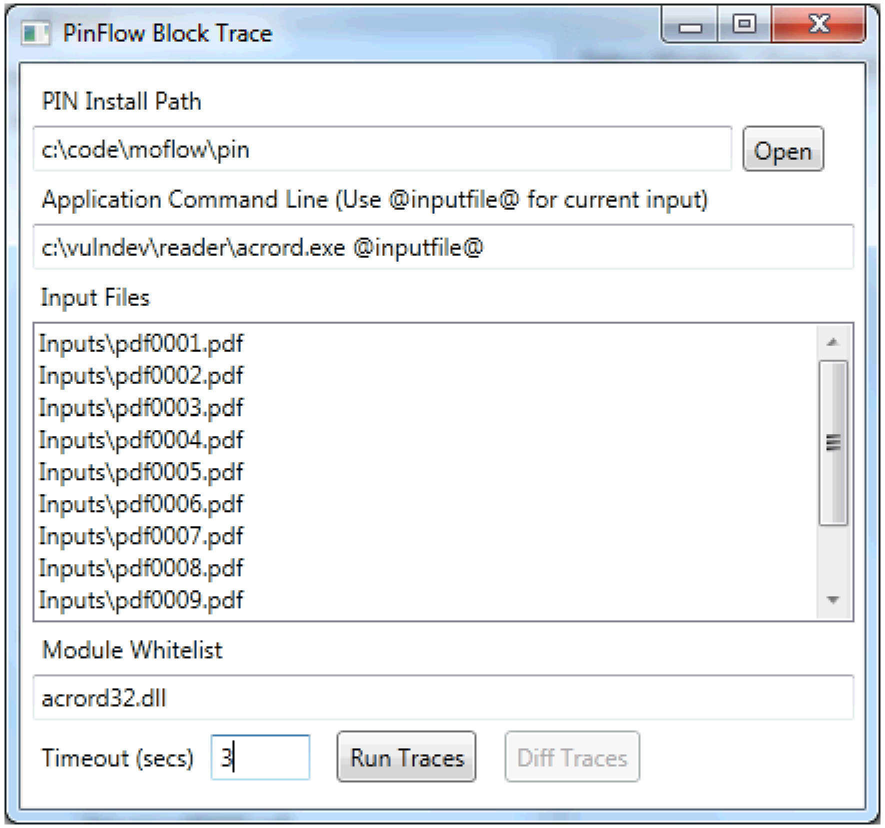
Template Code Coverage

- Dynamic Tracing
  - Instrument each basic block in a program
  - Efficiently record execution order of all blocks

- Implementation - PinFlow
  - Program tracer written as a PinTool
  - Hook on block cache creation
  - Inject instructions into cached code blocks
  - Callback function writes binary struct to ringbuffer
  - Ringbuffer flushed when full and on program exit

Template Code Coverage

- Moflow Visualizer PinFlow Trace Launcher



PIN Install Path
c:\code\moflow\pin

Application Command Line (Use @inputfile@ for current input)
c:\vulndev\reader\acrord.exe @inputfile@

Input Files
Inputs\pdf0001.pdf
Inputs\pdf0002.pdf
Inputs\pdf0003.pdf
Inputs\pdf0004.pdf
Inputs\pdf0005.pdf
Inputs\pdf0006.pdf
Inputs\pdf0007.pdf
Inputs\pdf0008.pdf
Inputs\pdf0009.pdf

Module Whitelist
acrord32.dll

Timeout (secs) 3

Template Code Coverage

- Advantage – Speed
  - PIN is much faster than traditional breakpoint or trap based solutions

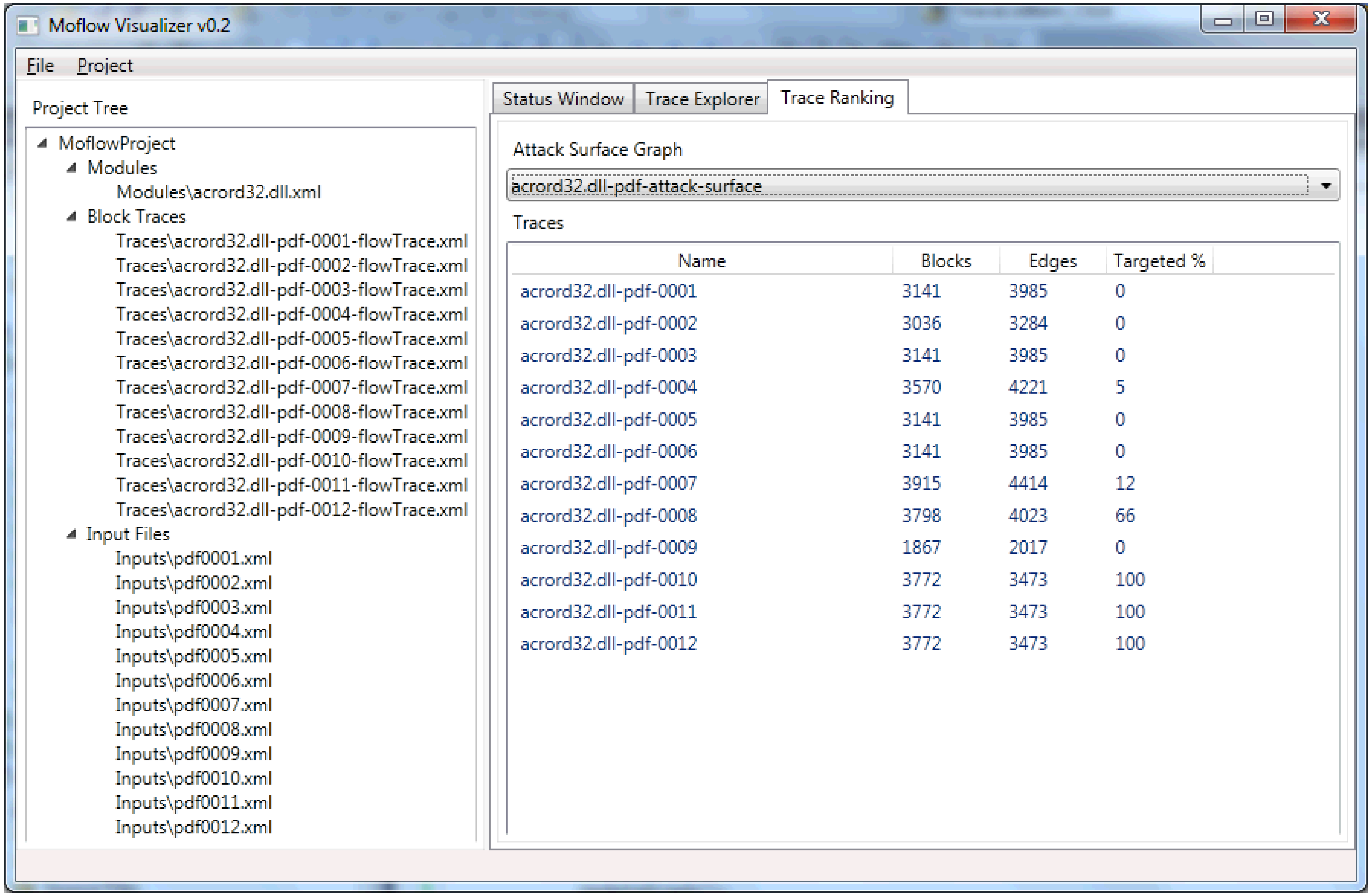
7zip Benchmark Test	
Block Tracer	Time (sec)
Process Stalker	20.48
PinFlow	1.77

11.57 times faster!

Template Prioritization

- Select functions for attack surface
- Calculate reachability to create attack surface graph
- Rank stored traces by number of nodes hit in attack surface graph

Template Prioritization



Project Tree

- MoflowProject
  - Modules
    - Modules\acrord32.dll.xml
  - Block Traces
    - Traces\acrord32.dll-pdf-0001-flowTrace.xml
    - Traces\acrord32.dll-pdf-0002-flowTrace.xml
    - Traces\acrord32.dll-pdf-0003-flowTrace.xml
    - Traces\acrord32.dll-pdf-0004-flowTrace.xml
    - Traces\acrord32.dll-pdf-0005-flowTrace.xml
    - Traces\acrord32.dll-pdf-0006-flowTrace.xml
    - Traces\acrord32.dll-pdf-0007-flowTrace.xml
    - Traces\acrord32.dll-pdf-0008-flowTrace.xml
    - Traces\acrord32.dll-pdf-0009-flowTrace.xml
    - Traces\acrord32.dll-pdf-0010-flowTrace.xml
    - Traces\acrord32.dll-pdf-0011-flowTrace.xml
    - Traces\acrord32.dll-pdf-0012-flowTrace.xml
  - Input Files
    - Inputs\pdf0001.xml
    - Inputs\pdf0002.xml
    - Inputs\pdf0003.xml
    - Inputs\pdf0004.xml
    - Inputs\pdf0005.xml
    - Inputs\pdf0006.xml
    - Inputs\pdf0007.xml
    - Inputs\pdf0008.xml
    - Inputs\pdf0009.xml
    - Inputs\pdf0010.xml
    - Inputs\pdf0011.xml
    - Inputs\pdf0012.xml

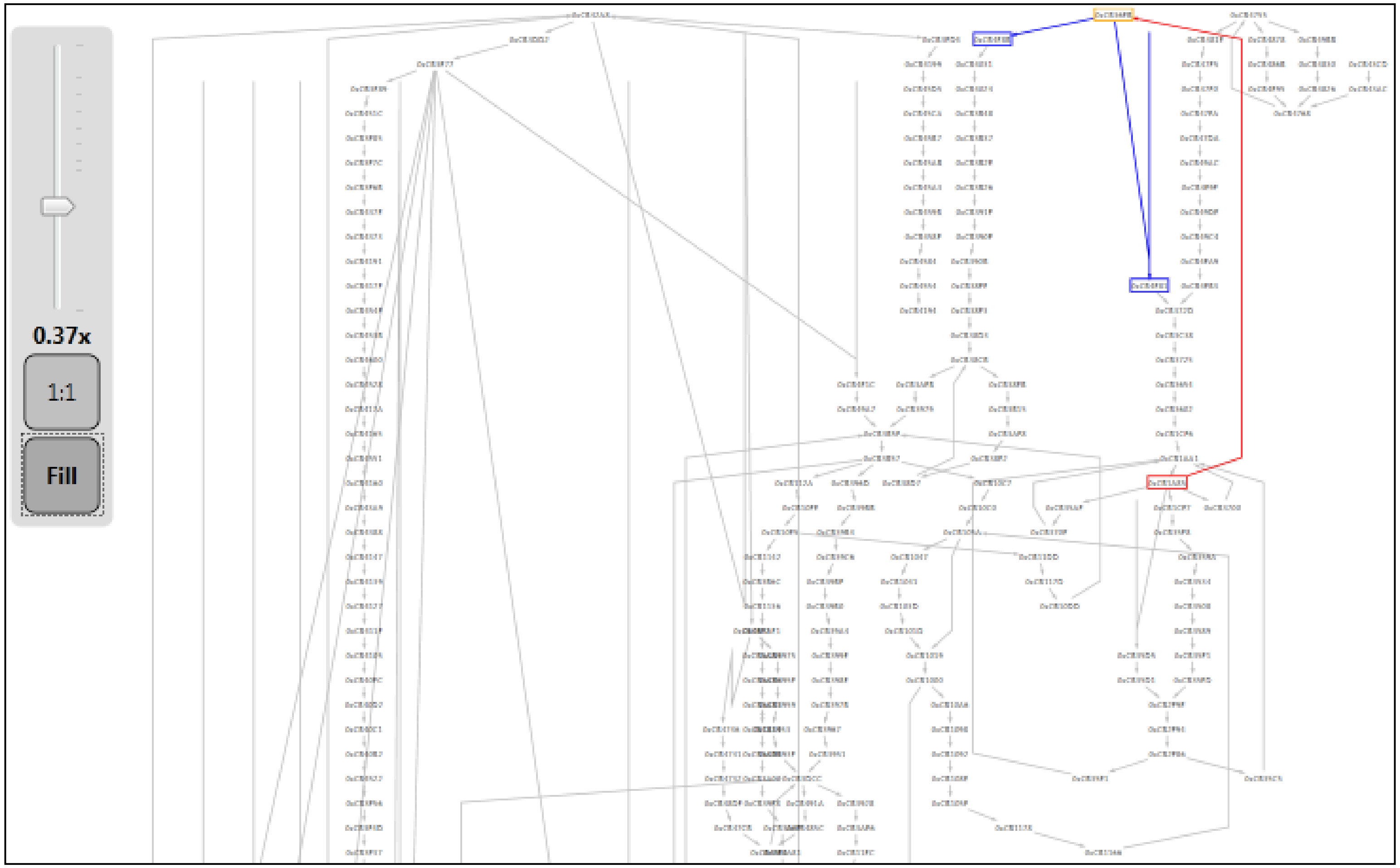
Attack Surface Graph: acrord32.dll-pdf-attack-surface

Traces

Name	Blocks	Edges	Targeted %
acrord32.dll-pdf-0001	3141	3985	0
acrord32.dll-pdf-0002	3036	3284	0
acrord32.dll-pdf-0003	3141	3985	0
acrord32.dll-pdf-0004	3570	4221	5
acrord32.dll-pdf-0005	3141	3985	0
acrord32.dll-pdf-0006	3141	3985	0
acrord32.dll-pdf-0007	3915	4414	12
acrord32.dll-pdf-0008	3798	4023	66
acrord32.dll-pdf-0009	1867	2017	0
acrord32.dll-pdf-0010	3772	3473	100
acrord32.dll-pdf-0011	3772	3473	100
acrord32.dll-pdf-0012	3772	3473	100

Graph Visualization

Moflow Block Trace Graph Visualization



Fuzzing Automation

Fuzzing Automation

- Distributed Fuzzing
- Fuzzer Management
- Data Gathering
- Crash Mining

Distributed Fuzzing

- Tests are small and atomic
  - Distribute simply
  - Make it easy to add systems
  - Easy to add tests

- Centralized Management
  - Aids in speedy addition of hardware

Fuzzer Management

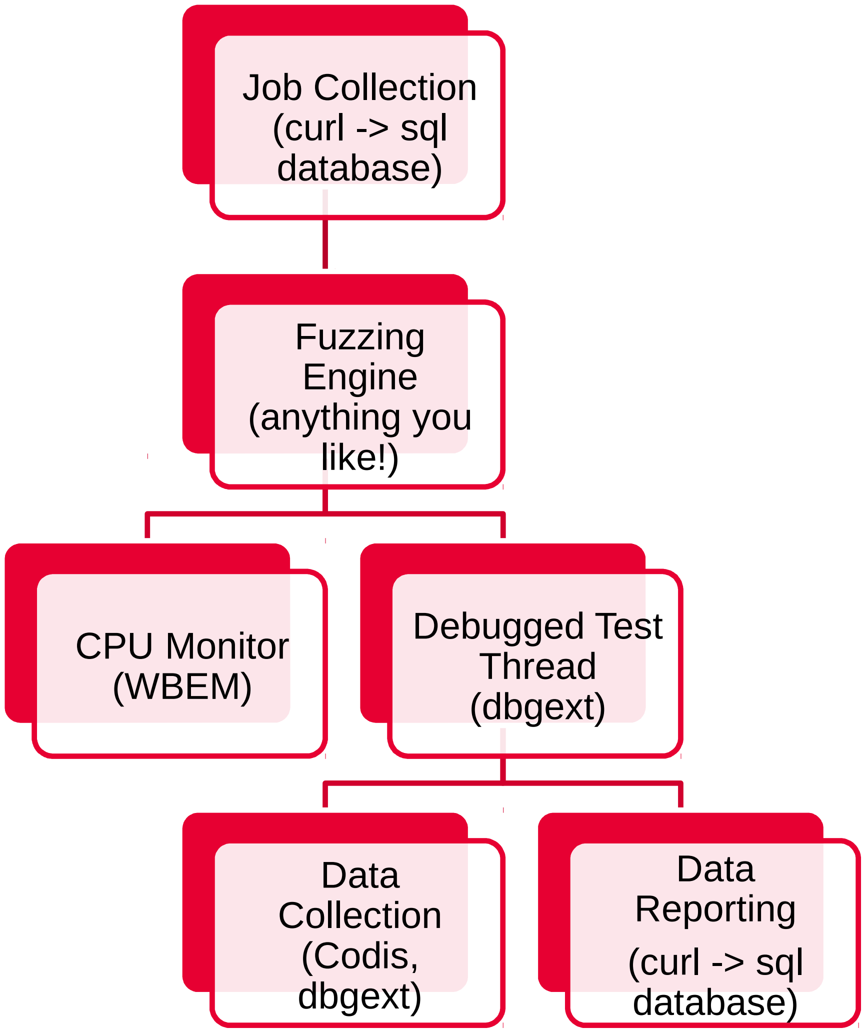
- Customizable yet simple
  - Ignore first chance exceptions?
  - Add debugging technologies?
  - Max test case timeout

- Ease of use is key
  - Quick recovery for dead hosts
  - Quick addition of new hosts
  - Centralized management w/ database

Fuzzer Management

- Jobs are held in the central DB
  - Job details passed to workers
  - Test cases are generated by workers as needed
  - Successful crashes are returned to the DB with details
- Test cases are wrapped with a custom debugger
- Data is returned to the central DB

Basic Worker


Job Collection (curl -> sql database)
Fuzzing Engine (anything you like!)
CPU Monitor (WBEM)
Debugged Test Thread (dbgext)
Data Collection (Codis, dbgext)
Data Reporting (curl -> sql database)

Data Gathering

- Store what you must
  - Bucketing
  - Categorization
  - Indicators of Exploitability

- Store what you have
  - Why redo work?
  - Can’t know what you may need

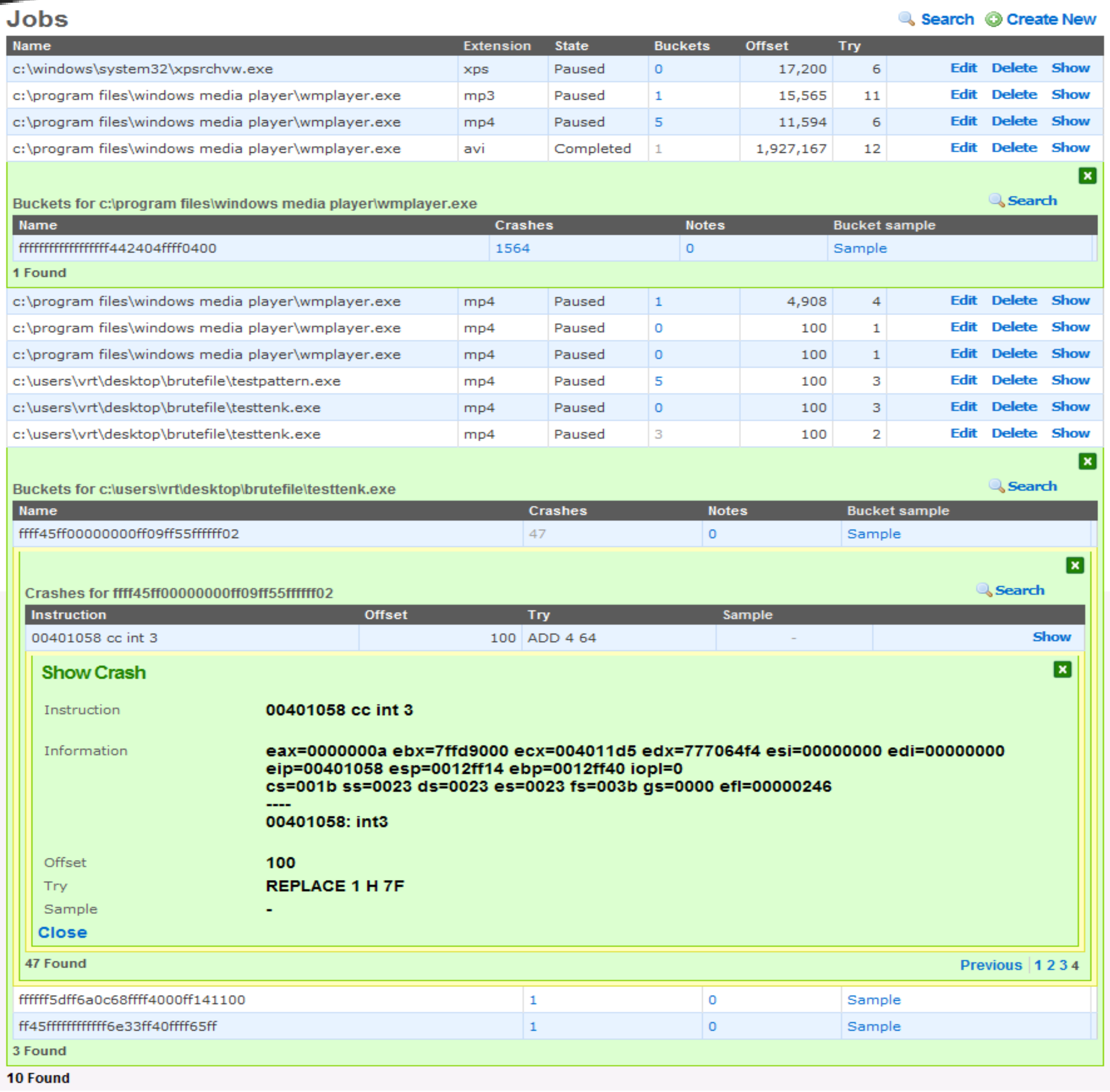
- Store it smart
  - Database!

Crash Mining

- Post-crash analysis is performed on crashes deemed "relevant"

- Relevant crashes are those which are:
  - Familiar to your exploit developers
  - Relate to your attacking goals

- Relevant crashes are mined as needed from the database with queries.
  - What is relevant changes over time.



Jobs

Search Create New

Name	Extension	State	Buckets	Offset	Try			
c:\windows\system32\xpsrchvw.exe	xps	Paused	0	17,200	6	Edit	Delete	Show
c:\program files\windows media player\wmplayer.exe	mp3	Paused	1	15,565	11	Edit	Delete	Show
c:\program files\windows media player\wmplayer.exe	mp4	Paused	5	11,594	6	Edit	Delete	Show
c:\program files\windows media player\wmplayer.exe	avi	Completed	1	1,927,167	12	Edit	Delete	Show

Buckets for c:\program files\windows media player\wmplayer.exe

Search

Name	Crashes	Notes	Bucket sample
ffffffffffffffff442404ffff0400	1564	0	Sample

1 Found

Name	Extension	State	Buckets	Offset	Try			
c:\program files\windows media player\wmplayer.exe	mp4	Paused	1	4,908	4	Edit	Delete	Show
c:\program files\windows media player\wmplayer.exe	mp4	Paused	0	100	1	Edit	Delete	Show
c:\program files\windows media player\wmplayer.exe	mp4	Paused	0	100	1	Edit	Delete	Show
c:\users\vrt\desktop\brutefile\testpattern.exe	mp4	Paused	5	100	3	Edit	Delete	Show
c:\users\vrt\desktop\brutefile\testtenk.exe	mp4	Paused	0	100	3	Edit	Delete	Show
c:\users\vrt\desktop\brutefile\testtenk.exe	mp4	Paused	3	100	2	Edit	Delete	Show

Buckets for c:\users\vrt\desktop\brutefile\testtenk.exe

Search

Name	Crashes	Notes	Bucket sample
ffff45ff00000000ff09ff55ffffff02	47	0	Sample

Crashes for ffff45ff00000000ff09ff55ffffff02

Search

Instruction	Offset	Try	Sample	
00401058 cc int 3	100	ADD 4 64	-	Show

Show Crash

Instruction	00401058 cc int 3
Information	eax=0000000a ebx=7ffd9000 ecx=004011d5 edx=777064f4 esi=00000000 edi=00000000 eip=00401058 esp=0012ff14 ebp=0012ff40 iopl=0 cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000246 ---- 00401058: int3
Offset	100
Try	REPLACE 1 H 7F
Sample	-

```
eax=0000000a ebx=7ffd9000 ecx=004011d5 edx=777064f4 esi=00000000 edi=00000000
eip=00401058 esp=0012ff14 ebp=0012ff40 iopl=0
cs=001b ss=0023 ds=0023 es=0023 fs=003b gs=0000 efl=00000246
----
00401058: int3
```

Close

47 Found

Previous 1 2 3 4

Name	Crashes	Notes	Bucket sample
ffffff5dff6a0c68ffff4000ff141100	1	0	Sample
ff45fffffffffff6e33ff40ffff65ff	1	0	Sample

3 Found

10 Found

Moflow: Triage

Triage - Requirements

- Exploitability
  - Exception information
  - Deep Trace
- Triggering Condition
  - Fuzzer feedback
  - Taint analysis
- Root Cause
  - Graph analysis

Triage - Exploitability

- Exception Information
  - Brutefile outputs XML data containing exception information

- Deep Trace
  - Code Coverage
  - Attack surface APIs
  - Dataflow

Triage - Exploitability

- Dataflow
  - Once exception is found program is traced using PinFlow to gather instruction level instrumentation
  - Blocks are hooked during cache and disassembled to instrument instructions that access memory
  - Dataflow callback function records the address and value of each memory read or write

- Taint Analysis
  - Provides exception analysis functions with information about controlled bytes
  - Knowledge of controlled bytes allows more precise analysis

Triage – Triggering Condition

- Fuzzer Feedback
  - As part of exception analysis data Brutefile includes information about mutation
- Taint analysis
  - When triaging a bug from input with unknown modifications, perform taint analysis
  - Forward taint propagation from memory allocated to stored data from input file will reveal which bytes are referenced in the exception

Triage – Root Cause

- Graph Analysis
  - ► Overlay graphs of several deep traces to determine similarity
  - ► If execution trace leading up to the crash is identical but different bytes were manipulated, root cause should be determined

- Taint analysis
  - ► Follow tainted data in the exception back to the code location that first influenced the memory location with external data

Moflow: Tools

Console Disassembler

- Console interface for libcodis

- Static Analysis
  - Instruction Disassembly
  - Function Detection
  - Code and Data Cross-References
  - Function Control Flow Graph
  - Call Graph

- Import IDA2Moflow and .map files

Windbg Integration

- CodisExt
  - Windbg extension using the engextcpp API
  - Utilizes libcodis to extract disassembly graphs and cross-references
  - Utilizes Windbg DML functionality to allow a hyperlinked interface for cross references

Windbg Integration

```
0:000> !codis
[codis] Usage:
[codis] !codis load <moduleName>                      Load a module into the
disassembler engine
[codis] !codis xrefs [functionAddr]                   Show caller/callees
[codis] !codis callers <functionAddr>                 Show function callers
[codis] !codis callees <functionAddr>                 Show function callees
[codis] !codis names                                  Show names in codis database
[codis] !codis dis <moduleName> [functionAddr]        Dump disassembly of a module or
function
[codis] !codis dot                                    Dump a GraphViz DOT file

0:000> !codis load test
[codis] Loading C:\Vulndev\test.exe

;------------------------------------------------------------------------------
; File Header
;------------------------------------------------------------------------------
; Binary format: 32-bit PE
; Byte Ordering: Little Endian
; Entry Point:   0000130b
; File Size:     112128 bytes
;------------------------------------------------------------------------------
```

Windbg Integration

```
0:000> !codis xrefs
[codis] Function: 00401005 sub_00401005
[codis] xrefs to: 00401149
[codis] xrefs from:
[codis] Function: 0040100a sub_0040100a
[codis] xrefs to: 0040100f
[codis] xrefs from:

--- SNIP ---

[codis] Function: 00411850 sub_00411850
[codis] xrefs to: 00411763
[codis] xrefs from:
[codis] Function: 00411a58
wrapper_RtlUnwind
[codis] xrefs to: 0040e530 00407732
[codis] xrefs from:
[codis] Function: 44cbe836 sub_44cbe836
[codis] xrefs to: 0040e53
```

```
0:000> !codis dot
digraph G {
"00401005"
"0040100a"
"0040100f"
"004010c0"
"0040113a"

--- SNIP ---

"00401076" -> "0040100a"
"00401058" -> "0040113a"
"0040104b" -> "004010c0"
"0040100f" -> "00401070"
"0040100a" -> "00401030"
}
```

Windbg Integration

```
0:000> !codis dis test 00402eea
00402eea |
........ |  ;;;;;;;;;;;;;;;;;;;;;;;;;;;
........ |  ;;; S U B R O U T I N E ;;;
........ |  ;;;;;;;;;;;;;;;;;;;;;;;;;;;
........ |  sub_00402eea:                                ;  xrefs:  0x00402f68  0x00402f31
0x004015f7
........ |  6a 08                    |    push byte  0x8                          ;
00402eec |  68 60 a2 41 00           |    push dword  0x41a260                    ;
00402ef1 |  e8 32 fd ff ff           |    call    <sub_00402c28>                  ;
00402ef6 |  e8 2e f9 ff ff           |    call    <sub_00402829>                  ;
00402efb |  8b 40 78                 |    mov     [eax+0x78], eax                 ;
00402efe |  85 c0                    |    test    eax, eax                        ;
00402f00 |  74 16                    |    jz      0x402f18                        ;
00402f02 |  83 65 fc 00              |    and dword  0x0, [ebp-0x4]               ;
00402f06 |  ff d0                    |    call    eax                             ;
00402f08 |  eb 07                    |    jmp     0x402f11                        ;
00402f0a |  33 c0                    |    xor     eax, eax                        ;
00402f0c |  40                       |    inc     eax                             ;
00402f0d |  c3                       |    ret                                     ;
00402f0e |  8b 65 e8                 |    mov     [ebp-0x18], esp                 ;
00402f11 |
........ |  loc_00402f11:                                                         ;  xrefs:
0x00402f08
........ |  c7 45 fc fe ff ff ff     |    mov dword  0xfffffffe, [ebp-0x4]        ;
00402f18 |
........ |  loc_00402f18:                                                         ;  xrefs:
0x00402f00
........ |  e8 46 48 00 00           |    call    <sub_00407763>                  ;
```

IDA Integration

- IDA2Moflow.idc
  - ▸ Dumps static program call graph
    - Module
    - Functions
    - Calls
  - ▸ Works on all versions of IDA

- Useful to overcome current limitations in static analysis provided by libcodis

Questions?

- Email: rjohnson@sourcefire.com
  richinseattle@gmail.com
- Twitter: Richinseattle

- Email: lgrenier@sourcefire.com
  pusscat@metasploit.com
- Twitter: Pusscat

- Special Thanks to Chris McBee!

Extra Slides

Template Code Coverage

- Dynamic Tracing

- Implementation
  - Program tracer written as a PinTool
  - Designed for Win32 platform
  - Function and Block hooking for Code Coverage
  - System call hooking for I/O*
  - Memory reference trace*
  - Logging to standardized format

Static Analysis

- Instruction Disassembly
- Function Detection
- Code and Data Cross-References
- Function Control Flow Graph
- Module / Program Call Graph

Instruction Decoding

- BeaEngine 4
  - ▸ Multi-Architecture
    - x86 / x64
  - ▸ High performance
    - [stats]
  - ▸ Actively developed
    - [stats]

Function Detection

- Prologue Detection
  - [Image of prologues]

- Static call targets
  - [show dynamic call vs static call]

Code and Data Cross-References

- Disassembly of functions results in extraction of CALLs, JMPs, and static data references

- [image goes here]

Function Control Flow Graph

- Break a function into basic blocks
  - JMP
  - CALL
  - RET

Module / Program Graph

- Enumerate function cross references

- Support loading multiple modules for inter-modular call graph

Dynamic Analysis

- LibMoflow
  - High level program analysis library in C#

  - Code Coverage Analysis
  - Trace Differencing
  - Graph Analysis
  - Tainted Data Analysis

Code Coverage Analysis

- Augment graph from static analysis with code coverage
- Trace Differencing
- CrashViz
  - Program Graph
  - Trace Overlays

Trace Differencing

- Describe algorithm here

Graph Analysis

- Loop Detection
  - Dominator Trees
  - etc

File Visualization

- Hex and Structutred Tree Views
- Visualize Fuzzer File Mutations and other session metadata
- Structure Decoding
  - Office Formats (GUT)
  - PDF (Only’s lib?)
  - FLASH (Patrick/Shong’s lib)