Accurate, Low Cost and Instrumentation-Free Security Audit Logging for Windows

Shiqing Ma, Kyu Hyung Lee, Chung Hwan Kim, Junghwan Rhee, Xiangyu Zhang, Dongyan Xu (ACSAC '15)

Presented by Noor Michael

CS 563 (Fall 2018)

ILLINOIS

Motivation

- Forward/Backward Tracing from Detected Anomalies
- Abnormal Causal Dependencies between Applications
- Track Information Flow

Windows ETW

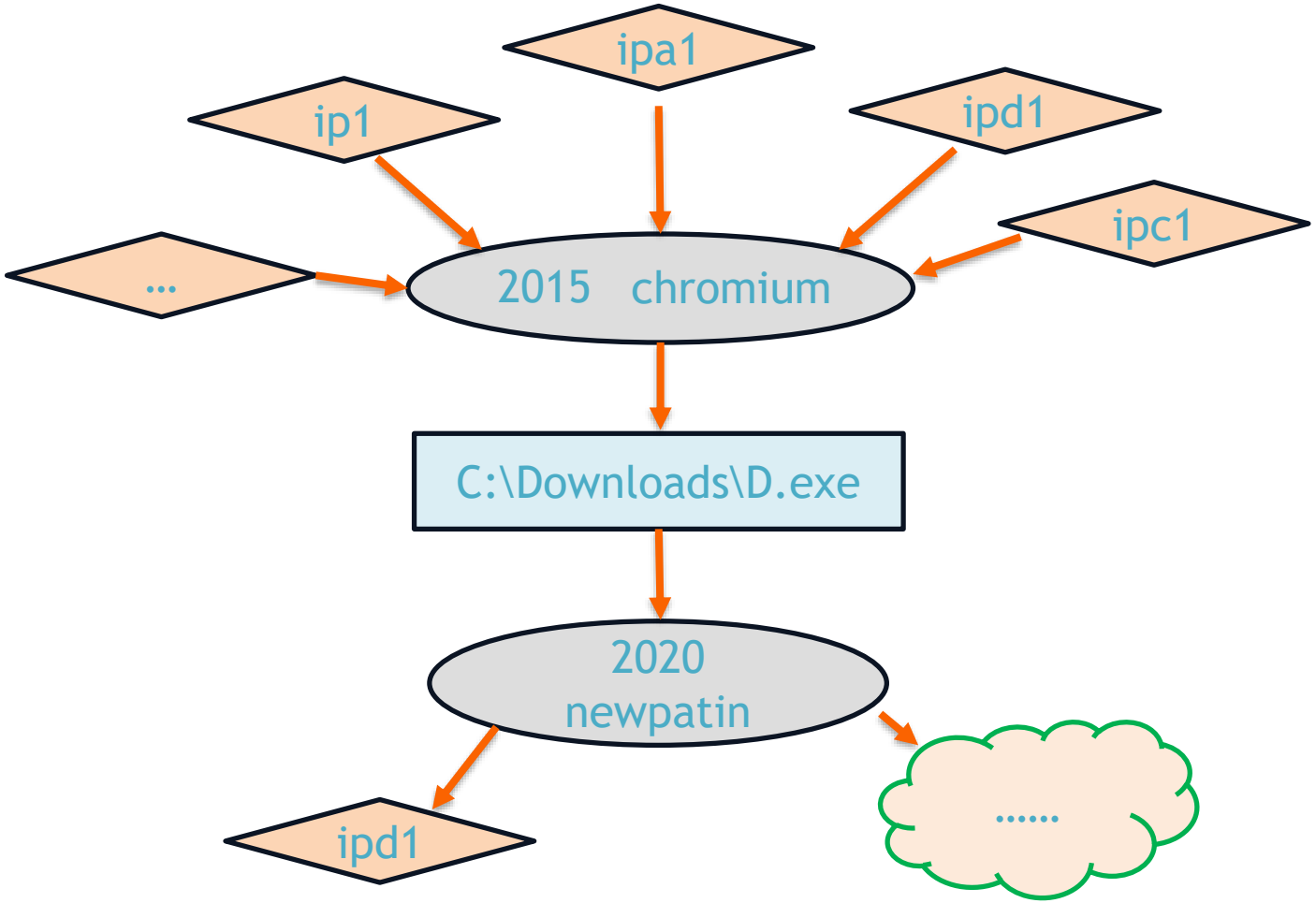
Event(TimeStamp,Processor)

- Event Type
  - FileRead etc.
- Event specific
  - FileObject, IOFlags etc.
- Process ID, Thread ID etc.
- Stack

Stack

1. TurboDispatchJumpAddressEnd+0x690@wow64cpu.dll
2. …
3. winnt_get_connection+0x4b@libhttpd.dll
4. worker_main+0x27@libhttpd.dll
5. …
6. RtlInitializeExceptionChain+0x36@ntdll.dll

Provenance Graphs



Execution Partitioning

- Parse Log File into Execution Units (Event Loops)
- Remove Intra-Unit Dependencies
- Output Events causally related to Input Events in same Unit
- Reduces Complexity and Overhead of Provenance Graphs

Methodology

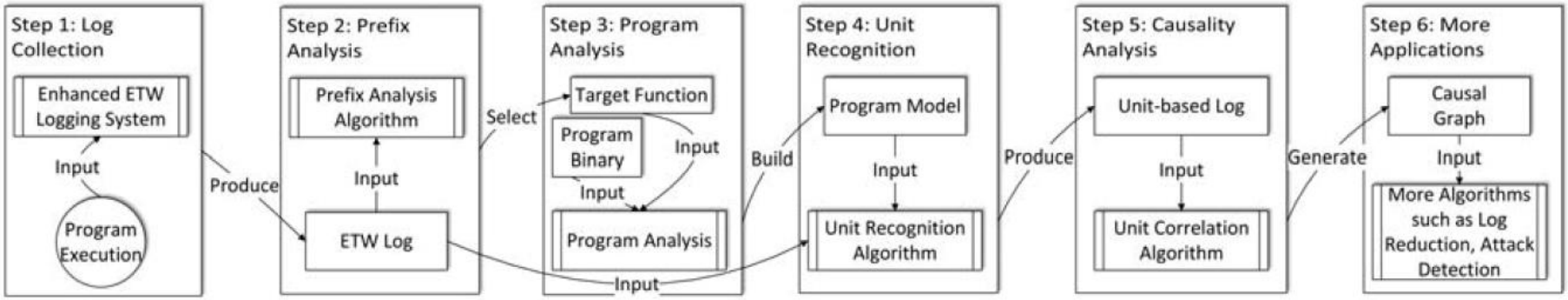


Figure 2: Approach Overview.

Log Collection

- ETW Collects Execution Log for Commonly used Applications
- Extend ETW to record Memory Operations (Clipboard Buffer)

```
Timestamp:      0x20462fbd1fb                    (1)
EventType:      WinsockTcpIpReceive
-Event Detailes-------------------------------------
TransferSize:   8000                             (2)
-Process--------------------------------------------
Process:        3112 httpd.exe                   (3)
Thread:         3948 msvcrt.dll!endthreadex+0x29
-Stack Trace----------------------------------------
  ...(Library and kernel functions)              (4)
  libapr-1.dll!apr_socket_recv+0x42
  ...(Application functions)
  libhttpd.dll!worker_main+0x9c
  ...(Kernel functions)
```

Prefix Analysis

- Identify *Target Function*, which contains Event Processing Loop
- Execution: Prologue, Event Handling Phase, Epilogue
- Prologue and Epilogue same between Executions

Prefix Analysis

```
1.  void main() {
2.      init();
3.      while(True) {
4.          read_cmd();
5.          if (cmd == FileDownload) {
6.              if(file ready) {
7.                  fd = open_file(file_name);
8.                  if(open fails)
9.                      errmsg_continue(MSG2);
10.                 buf = memory_allocation(size);
11.                 while(transfer not done) {
12.                     read_file(fd, buf);
13.                 write-data(socket, buf);
14.                 }
15.                 memory_free(buf);
16.                 close_file(fd);
17.             } else
18.                 errmsg_continuemsg(socket, MSG3);
19.         } else if(cmd ==...) { ... }
20.     }// end while
21.     server_exit();
22. }
```

```
SocketRead:   windows_runtime->main->read_cmd->...
FileOpen:     windows_runtime->main->open_file->...
FileRead:     windows_runtime->main->read_file->...
SocketWrite:  windows_runtime->main->write_data->...
FileClose:    windows_runtime->main->close_file->...
```

Model Construction

- Identify Event Handling Loop from PCs in Event Handling Logs
- Represent Possible Sequences of Function Calls in Loop by Regular Expression
- Recursively Disassembles Callee Functions until reaches sufficiently Informative Model
- Use ETW Logs if Binary too Complex

Model Construction

```
1.  void main() {
2.      init();
3.      while(True) {
4.  F1:     read_cmd();
5.          if (cmd == FileDownload) {
6.              if(file ready) {
7.  F2:             fd = open_file(file_name);
8.                  if(open fails)
9.  F3:                 errmsg_continue(MSG2);
10.                 buf = memory_allocation(size);
11.                 while(transfer not done) {
12. F4:                 read_file(fd, buf);
13. F5:                 write_data(socket, buf);
14.                 }
15.                 memory_free(buf);
16. F6:             close_file(fd);
17.             } else
18. F7:             errmsg_continuemsg(socket, MSG3);
19.         } else if(cmd ==...) { ... }
20.     }// end while
21.     server_exit();
22. }
```

Model(3-20)$

=F1•Model(5-20)$

=F1•(Model(6-18) | …)$

=F1•((Model(7-16) | F7)| …)$

=F1•((F2•Model(8-16) | F7)| …)$

=F1•((F2•([F3] •Model(11-16))|F7)|…)$

=F1•((F2•([F3] •(Model(11-14)•F6))|F7)|…)$

=F1•((F2•([F3] •((F4•F5)*•F6))|F7)|…)$

Log Partitioning

- Separate Logs by Process ID
- When end of sequence is reached, begin parsing another unit
- Can remove library functions which never lead to syscalls
- Always parse longest possible subsequence of events as unit

Dependency Analysis

- Construct Causality Graph between Units and System Objects
- Output Events are only dependent on Input Events of the same Unit
- Prune Dependency-Free Objects
- Prune Repeated Reads/Writes

Evaluation

- ETW Log overhead of ~10-18% for high workload
- Around ~0.4-2.5% overhead for normal workload

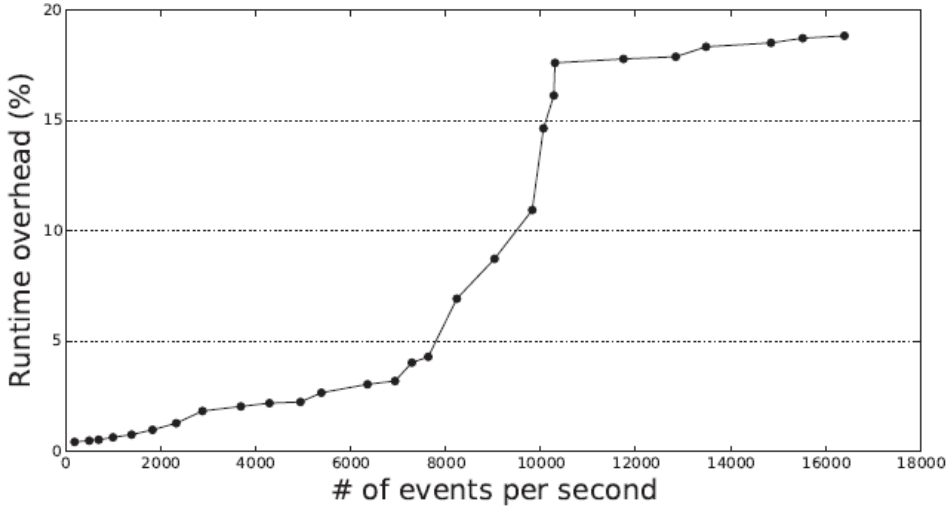


Figure 8: Event Tracing Overhead.

Evaluation

Application	# of Events / Ratio				# of Units		# of Events per unit	
	Original	Dependency-free	Redundant	Final	Original	Final	Original	Final
TextTransfer	316	310 / 98.10%	0 / 0.00%	6 / 1.90%	2	2	158.00	3
Chromium	102,206	61,193 / 59.87%	36,834 / 36.04%	4,179 / 4.09%	255	248	400.81	16.85
DrawTool	15,438	13,382 / 86.68%	1,982 / 12.84%	74 / 0.48%	11	11	1,403.45	6.73
NetFTP	10,621	8,909 / 83.88%	1,132 / 10.66%	580 / 5.46%	8	8	1,327.63	72.5
AdvancedFTP	1,615	1,161 / 71.89%	411 / 25.45%	43 / 2.66%	3	3	538.33	14.3
Apache httpd	37,171	27,035 / 72.73%	8,084 / 21.75%	2,052 / 5.52%	46	46	808.07	44.61
IE	29,969	12,983 / 43.32%	14,711 / 49.09%	2,275 / 7.59%	10	10	2,996.90	227.5
Paint	7,085	6,772 / 95.58%	235 / 3.32%	78 / 1.10%	28	28	253.04	2.78
Notepad	11,704	8,506 / 72.68%	3,168 / 27.07%	30 / 0.26%	6	6	1,950.67	5.00
Notepad++	5,516	4,976 / 90.21%	404 / 7.32%	136 / 2.47%	9	9	612.89	15.11
SimpleHTTP	779	559 / 71.76%	180 / 23.11%	40 / 5.13%	13	13	59.92	3.08
Sublime Text	30,372	24,419 / 80.40%	5,637 / 18.56%	316 / 1.04%	11	11	2,761.09	28.73

Table 3: Effectiveness of Log Reduction.

Evaluation

Scenario	# of events			# of nodes			# of edges			Correctness	
	Before	After	Ratio	Original	Unit	GC	Original	Unit	GC	Backward	Forward
Mis-configured server	986,563	85,042	8.62%	173	10	10	204	10	10	-	Match
Phishing attack	523,385	44,593	8.52%	573	21	21	693	32	32	Match	Match
Information leak	1,947,485	260,857	13.39%	10,222	11	11	20,532	10	10	Match	Match
Spyware	1,284,748	102,523	7.98%	9,282	9	9	11,244	8	8	Match	Match

Table 4: Attack scenarios summary with original log and reduced log.

Recap

- Especially useful for Network Services
- 12X – 95X Space Reduction with Garbage Collection

Discussion

- Assumptions (Limitations) of the Paper
- Other Program Analysis Techniques to improve Model
- Other Applications of this Methodology

Comments

- Causal Relationships between Units (e.g. Browsers)
  - Attackers Distribute Effects across Units
- Attackers Use Temporary Files (Dependency-Free Objects)
- Evaluated Performance, but not Effectiveness in Analysis
- Extensive use of Heuristics