DAMM: Differential Analysis of Malware in Memory

Dr. Vico Marziale
Managing Partner

OSDFC 2014

#id

- Managing Partner @504ENSICS Labs in NOLA
- PhD in CS from UNO
  - Briefly taught security/crypto stuff
- Many hats:
  - R&D
  - Penetration testing
  - Malware analysis
  - Digital Forensics
- Contributor to/developer of: Registry Decoder, Scalpel, Spotlight Inspector, DAMM
- Co-organizer BSidesNOLA
- Tequila enthusiast

Malware Analysis: Hard!

- That is a problem. Do I get a cookie?
- Multiple types of analysis
  - Static (look at the binary)
  - Dynamic (run the binary and look)
  - Memory Analysis (a hybrid approach)
- Pros and cons re:
  - Time/expertise required
  - What malicious activity can be found
  - How malware can make life difficult
- Goal: Make memory analysis a bit easier. Cookie now?

Static Analysis

- Tools: disassemblers, decompilers, string/grep
- Pros:
  - Malware that's not running can't actively mess with you
    - Hiding processes
  - Or infect your network
  - Can see all environment based possible actions
- Cons:
  - Also can't unpack itself, decrypt itself, or perform network activity (download files, C2 communications)
  - Disassembly/reversing is hard (takes time, expertise)
  - Can't directly see effects on entire system

Dynamic Analysis

- Tools: debuggers, sandboxes, procmon, fakenet
- Pros:
  - Must be unpacked and decrypted to execute
  - Can see network activity
  - Less need for disassembly/reversing
  - Can potentially see entire running system
- Cons:
  - Peek-a-boo - it can (probably) see you
    - Whether debugger, host-based analysis, VM, or sandbox
  - Then lie to you (_EPROCESS linked lists ++)
  - And/or alter its behavior: sleep, exit, crash box, migrate via speaker/microphone (maybe …)
  - And perform malicious activities

Memory Analysis

- A bit like each of the previous
- Run the malware (host, VM, sandbox)
  - Like dynamic analysis
- Wait a sec (min, hour, day)
  - Some malware is sleepy
- Make copy or copies of memory
  - Bit for bit copy (snapshot) of physical RAM
- Analyze snapshot(s)
  - Like static analysis

Memory Analysis

- Pros:
  - Malware unpacked/unencrypted in memory
  - Network activity occurs
  - Malware captured in snapshot can’t mess with you
  - We get to relax and look at the entire running state of the system
- Win, right?

Problems

- Tons of Stuff
  - Have a 256GB image of RAM. Now what? Volatility!
- Great: parses all the things
  - Processes, network connections, DLLs, modules, open handles to files, sockets, registry, mutexes, etc.
- Difficulty: parses all the things
  - 10s of processes, sockets, connection
  - Hundreds of DLLs, loaded modules, and services
  - Thousands of handles for
    - Files, sockets, mutants, registry keys

Problems

- Tons of Samples
  - Always more samples than analysis muscle
- What is malicious?
  - Most are stock Windows objects: might not need to focus initially on these
  - Or at least are made to look like stock Windows objects (lssass.exe): definitely need to look at these
- Automation the key (At least for triage)
- Many type of analysis are not easy to automate (IDA)
- Some types are easier (Cuckoo)
- How about for Volatility?
- It is *extensible* and *open source (and fun)*

DAMM Intro

- *D*ifferential *A*nalysis of *M*alware in *M*emory
- FOSS tool built on Volatility
- Initially funded by DARPA Cyber Fast Track
- Python
- Command line
- Windows centric so far (but not for long)
- Duplicates Volatility output for many plugins
  - ~30 Volatility plugins combined into ~20 DAMM plugins
- Not so interesting

DAMM Intro (Cont.)

- Can analyze multiple copies of RAM
  - E.g., clean versus (suspected) infected
- And highlight differences between them
  - New drivers, processes, etc.
  - Changes in above
- SQLite backed
- Smart type-aware filtering
- Issue warning of ‘suspicious’ artifacts
- Multiple output formats
- Library-ized: libdamm (parse stuff into objects)
- Data reduction, expert domain knowledge, friendly output, performance
- Beta-ish!

Use Cases

- In virtual infrastructure
  - Snapshot RAM at each boot
  - See changes from boot to current
  - Or look further back in time
- Non-virtual environments
  - Keep Gold Standard disk image?
  - Do same for memory image!
    - Or generate as needed
- Malware analysis sandboxes
  - Configure to take before and after memory snapshots

Basic Usage

#python damm.py –-profile WinXPSP2x86 –f mem.img –p processes

Lists interesting information about running and exited processes

Output clipped, also gives start time, exit time, invocation, number of threads and open handles, etc. (combines pslist, psscan, psxview, ...)

offset	name	pid	ppid	prio	image_path_name
0x25c8830	System	4	0	8	
0x225ada0	alg.exe	188	668	8	C:\WINDOWS\System32\alg.exe
0x2114938	ipconfig.exe	304	968	8	
0x2086978	TSVNCache.exe	324	1196	8	C:\Program Files\TortoiseSVN\bin\TSVNCache.exe
0x22df020	smss.exe	376	4	11	\SystemRoot\System32\smss.exe

Similar plugins exist for ~20 other types of objects: dlls, network connections …

DAMM: Performance

User can opt to store results in SQLite db:

#python damm.py --profile WinXPSP2x86 –f mem.img –p processes –db mem.db

- Makes re-parsing instant
- Can easily be shipped to other investigators
- Or serve as an archive
- Db includes some simple metadata
    - No more need for memory snapshot (for plugin stuff)
    - No more need to specify profile

DAMM: DB Query

To use the db:

#python damm.py –p processes –db mem.db

To see some of what is stored in the db:

#python damm.py –db mem.db –q

Profile:	WinXPSP2x86
Memimg:	cridex.vmem
Computername:	ACCOUNTING12
Plugins:	processes dlls injection …

Also all of the envars for the explorer process (systemroot++ for warns)

Question?

- How do we determine what is *the bad*?
- Idea: get a clean copy of RAM from same/similar machine
- Compare before and after to infer malicious activity
  - New running processes
  - New loaded modules
  - …
- How, though? Use *diff* and we’re done, right?

To Diff or Not to Diff?

- Have two memory snapshots
- Each has a set of objects
  - Processes, DLLs, network connections, drivers
- How do we determine:
  - What *uniquely identifies* an object? (PID? Name?)
  - Which objects exist in both copies?
  - Only in the infected?
  - In both but changed (or not)?
  - Do the changes matter?

What is a ‘Process’

- Our notion has set of attributes
    - Name
    - PID
    - PPID
    - Physical address
    - Start time
    - # handles, threads
    - …
- Same process on same boot of same machine?
    - Physical offset, pid, ppid, name?
- Different machines?
- Plain diff is simply not going to work

DAMM: Differential Analysis

- Use two memory snapshots
  - Before infection (or known clean)
  - After infection (or suspected of infection)
- Select plugin(s)
- Parse a set of objects from each snapshot into dbs
  - Processes, DLLs, etc. from Volatility into objects
  - Using shims. Belch.
- Generate differences
- View only
  - New objects in the ‘after’ or infected snapshot
  - Objects in both, but have changed
- Unique ID defaults set for same boot of same machine

Differencing Example

#python damm.py –p processes –db infected.db –diff clean.db

status	offset	name	pid	ppid	prio	threads	handles
New	0x217f650	wpabaln.exe	1184	624	8	1	58
New	0x2408a78	wuauclt.exe	1596	1008	8	7	172
New	0x2288a78	WORDPAD.EXE	320	1204	8	2	98
New	0x22d3c10	cmd.exe	972	1956	8	1	33
New	0x216d228	win32dd.exe	1120	972	8	1	22
Changed	0x223d6a0	VMWARESERVICE.E	1812	668	13	2->3	82->132
Changed	0x247bb28	EXPLORER.EXE	1956	1932	8	16	293->427

DAMM: Unique ID

- For processes, unique id defined as
  - pid, ppid, name, creation_time
- This so far works for same machine same boot
- What about when not in controlled sandbox environment?
- Like a diff with clean image from another execution
- Change the unique id
  - name, image_path_name, command_line?
  - Accounts for binaries in wrong places, and normal duplicate names: svchost

Unique ID Example (1)

Here the dbs were generated from 2 different WinXPSP2 images from different machines:

#python damm.py –db infected.db –p processes –diff clean.db

status	offset	name	pid	ppid	prio	threads	handles
New	0x4b5a980	VMwareUser.exe	452	1724	8	8	204
New	0x655fc88	VMUpgradeHelper	1788	676	8	5	100
New	0x6945da0	spoolsv.exe	1432	676	8	14	137
New	0x1122910	svchost.exe	1028	676	8	88	1395

Everything (except System) shows up as new. Not helpful.

Unique ID Example (2)

Here also the dbs were generated from 2 different WinXPSP2 images from different machines:

#python damm.py –db infected.db –p processes –diff clean.db –u name image_path_name command_line

status	offset	name	pid	ppid	prio	threads	handles
New	0x10c3da0	wuauclt.exe	1732	1028	8	7	178
New	0x69d5b28	vmtoolsd.exe	1668	676	8	5	218
Changed	0x1bcd0b8->0x1214660	System	4	0	8	56->61	->179
Changed	0x18b4648->0x1122910	svchost.exe	1080->1028	692->676	8	66->88	1140->1395

More useful output, we see changed processes again (13/28 new)

DAMM: Filtering

- Further reduce set of objects
- Filter on objects’ attribute value: pid 4242
  - Find all about some process
  - DAMM knows PIDs versus other integers
- String search: string evil.dll
  - Find all occurrences of a DLL name
  - DAMM knows which attributes to search
- Filtering can be based on exact matching or partial

#python damm.py --db infected.db –p X –filter pid:1180

processes

offset	name	pid	ppid	prio	image_path_name
0x4a4b5f8	lanmanwrk.exe	1180	1060	8	C:\WINDOWS\System32\lanmanwrk.exe

privileges

pid	filename	value	privilege	present	enabled	default
1180	lanmanwrk.exe	17	SeBackupPrivilege	TRUE		
1180	lanmanwrk.exe	23	SeChangeNotifyPrivilege	TRUE	TRUE	TRUE
1180	lanmanwrk.exe	20	SeDebugPrivilege	TRUE		
1180	lanmanwrk.exe	10	SeLoadDriverPrivilege	TRUE	TRUE	

handles

offset	pid	type	name
0x80f5c260	1180	File	…\Documents and Settings\Administrator\Desktop
0xe1621ec0	1180	Key	…\WINDOWS\CURRENTVERSION\RUN
0xe1dd8a70	1180	Key	…\WINSOCK2\PARAMETERS\NAMESPACE_CATALOG5
0xe10f5188	1180	Key	…\WINSOCK2\PARAMETERS\PROTOCOL_CATALOG9
0xe1e96d30	1180	Key	…\WINDOWS\CURRENTVERSION\INTERNET SETTINGS

And so on for DLLs, network connections, etc…

DAMM: Warnings

Automatically search for suspicious objects:

- Processes running from temp directories
- DLLs loaded from temp directories
- PE headers in injectable memory pages
- For core Windows processes: correct priority, parent-child relationship, binary path
- Hidden processes, dlls
- Mangled filenames for important processes
- MFT pf entries for suspicious processes
- From the Volatility cheat sheet, The Book, “Know Your Windows Processes or Die Trying,” and elsewhere

#python damm.py –db infected.db --warnings

- Code injection:
  - services.exe (pid: 668) has PE header in injection.
- Number of process instances:
  - lsass.exe (pid: 1928) has 3 instances. Only one instance should exist!
- Proper parent/child process relationships:
  - lsass.exe (pid: 1928) parent process expected: winlogon.exe, actual: services.exe.
- Boot time processes starting long after boot:
  - lsass.exe (pid: 868, "C:\WINDOWS\\system32\\lsass.exe") started 18703082.0 seconds after boot time which may be suspicious.
- Process priority:
  - lsass.exe (pid: 1928) base priority expected: 9, actual: 8.
- Process unlinking:
  - 1_doc_RCData_61 (pid: 1336) may be a hidden process.
- Prefetch entries for suspicious processes
  - File: [MFT FILE_NAME] WINDOWS\Prefetch\REG.EXE-0D2A95F7.pf is a prefetch entry for a suspicious process.
- Mangled names:
  - winninit.exe ((pid: 4792) is named suspiciously similarly to a Windows process: wininit.exe.

Output Formats

To further ease analysis, output (using db) can be:

- TSV (for Excel or whatever, use –tsv)

- Grepable (if filtering doesn't suffice, use --grepable)

dlls: process_id: 584   process_name: csrss.exe dll_base: 0x75b50000   load_count: 0x3

dlls: process_id: 584   process_name: csrss.exe dll_base: 0x75b60000   load_count: 0x2

dlls: process_id: 584   process_name: csrss.exe dll_base: 0x77dd0000   load_count: 0x5

dlls: process_id: 584   process_name: csrss.exe dll_base: 0x7e720000   load_count: 0x1

- Screen formatted (the default, use 'less –S' or equivalent)

Conclusion

- That’s about it for now.
- Next up?
  - More warnings
  - Full Windows support
  - Then maybe Linux and Mac
  - More funding?
- Lots of things to think about in light of the presentations from yesterday and today

Questions?

Dr. Vico Marziale

vico@504labs.com

@vicomarziale

https://github.com/504ensicsLabs/DAMM