Fuzzing Memory Forensics Frameworks

Andrew Case

Arghya Kusum Das

Seung-Jong Park

J. Ramanujam

Golden G. Richard III

Research Motivation

- The number of artifacts recovered has to led a surge in analysis automation
  - DAMM
  - Evolve
  - VolDiff
  - VolUtil
  - … many more
- The effects of smear (and malware) lead to issues in automation

Smear & Hostile Environments

- *{memory, page, data} smear* is the term used to describe incorrect or missing information in a memory capture due to non-atomic acquisition
- Malware can tamper with the acquisition process and insert/modify/remove data
  - Rarely seen in the wild, many research POCs available

Fuzzing to the Rescue?

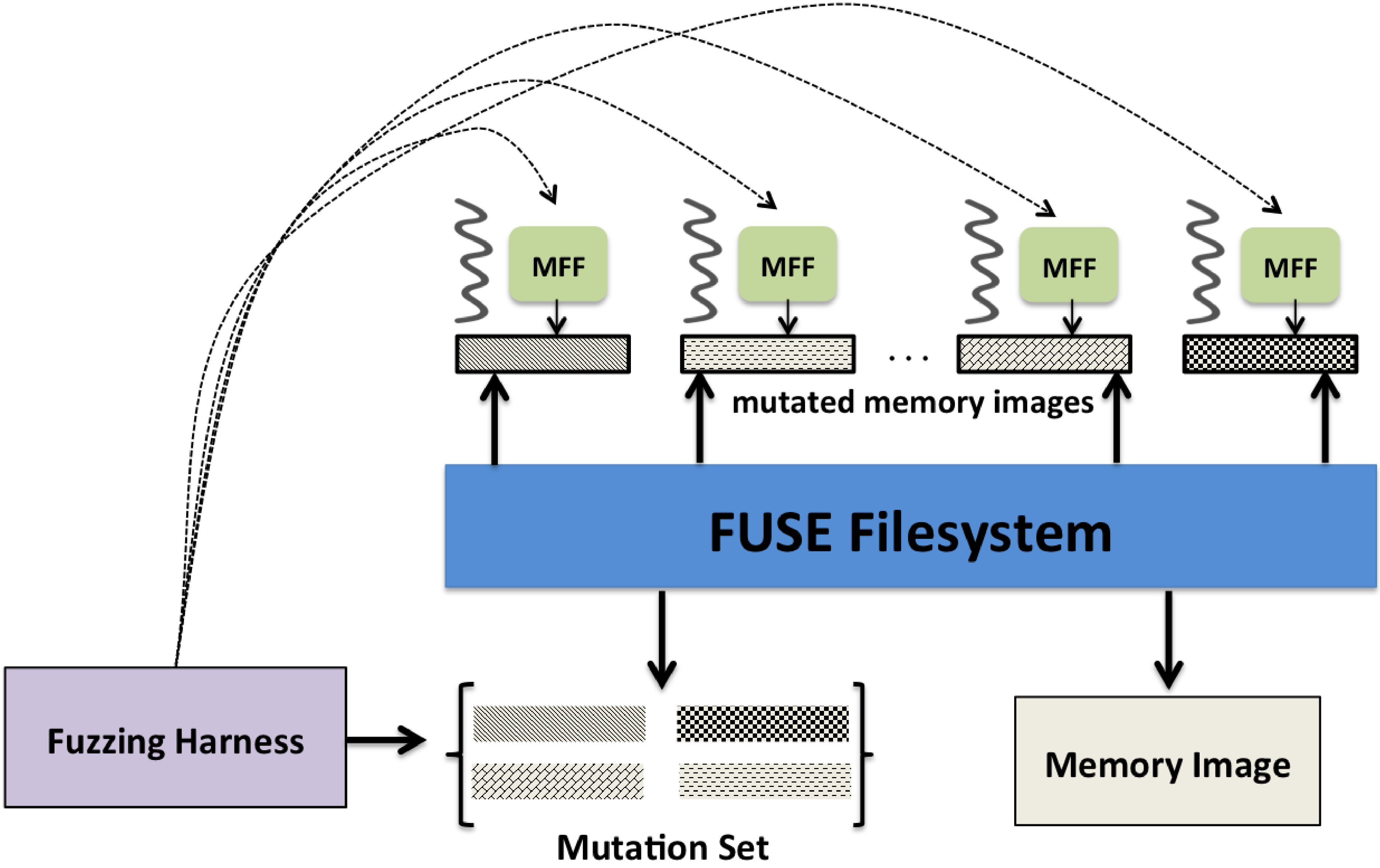
- Fuzzing is the technique of generating random (or semi-random) inputs in order to cause an application to crash or behave incorrectly
- Widely used in the application security testing industry
- Highly effective at uncovering programming errors in an automated fashion

Fuzzing Architecture Goals

1. Support all memory forensic frameworks (MFF), whether open or closed-source
2. Require no changes to the source code of the tested MFF
3. Only test the MFF portions that process memory samples
4. Only mutate the pages (bytes) of a memory sample processed by the MFF

Fuzzing Architecture Goals Cont.

5. Apply mutations dynamically at runtime to avoid making copies of tested samples
6. Test for a variety of programming and runtime errors
   - Inelegant crashes (no exception handling)
   - Infinite (or nearly infinite) loops
   - File-system resource exhaustion
   - Memory resource exhaustion


MFF
MFF
MFF
MFF
. . .
mutated memory images
FUSE Filesystem
Fuzzing Harness
Mutation Set
Memory Image

Mutations – Whole Page Smear

- Mutate entire page of data with 0x00, 0xff, and random bytes
- This closely mimics the effects of smear across entire pages that MFFs process

Mutations – Data Structure Smear

- Type 1 - On boundaries of 2, 4, 8, and 128 bytes, fill just the boundary byte with 0x00, 0xff, and a random value
- Type 2 - On the same boundaries and with the same values, fill the entire amount (2, 4, 8, or 128)
- Mimics data structure smear, which is when a data structure is partially overwritten and only partially valid

Mutations – Pointer Smear

- Similar to data structure smear, but the values are +/- 2, 4, 8, 128, or 4096 from the current value
- These mimics pointers being smeared to reference valid addresses, but invalid corresponding data

Target #1 - Volatility

- Widely used memory forensic framework
- We chose to fuzz 10-12 plugins per operating system (Windows, Linux, and OS X)
- Plugins were chosen based on popularity / usage frequency

Volatility - Crashes

Plugin Name	**Programming Error**
linux_library_list	List enumeration code didn't properly validate pointer to data structures before processing it
linux_dmesg	Did not validate that *log* structures referenced a valid page before attempting to process them
linux_arp	Integer overflow in bit shifting operation
mac_check_syscall	Crashed when system call table entries were not on a mapped page

Volatility – Infinite Loops

Plugin Name	**Programming Error**
linux_bash	List enumeration code infinite looped when a mutation caused two list members to point to each other
linux_arp	List enumeration code infinite looped when a mutated list entry pointed to a previous entry
mac_lsmod	List enumeration code infinite looped when a mutated list entry pointed to a previous entry
mac_lsof	Nearly infinite loop when the variable specifying how many handles a process had opened was mutated to ~3 billion

Volatility – Unusable Output

Plugin Name	**Programming Error**
linux_psaux	The *mm_struct* members that specify the start and end of the command line arguments were mutated to specify a size in the gigabytes
linux_psenv	Same as issue as *linux_psaux*, but for the members that specify the size of the process' environment variables
mac_dyld_maps	List enumeration code infinite looped and the rendering code did not validate structure properly before reporting (printing)
mac_psaux	Same base issue as *linux_psaux*

Volatility – File System Resource Exhaustion

Plugin Name	**Programming Error**
linux_procdump	Gaslight forced the plugin to attempt to create a 20GB+ file as the function used for extraction to disk, *write_elf_file*, did not properly validate file size metadata
linux_librarydump	This plugin relied on the same *write_elf_file* function when extracting shared libraries to disk

Volatility Windows Issues?

- No new issues in Windows plugins were found during our (limited) testing
- We tested Gaslight against a recently closed *vaddump* issue, and with the patch removed, Gaslight automatically triggered the bug
  - https://github.com/volatilityfoundation/volatility/issues/333

Target #2 – Rekall

- Due to time constraints, only tested Rekall's *arp* Linux plugin
- Gaslight uncovered three issues in the Rekall version
    1. Infinite loop in list walking code
    2. Nearly infinite loop in entry enumeration
    3. Insufficient checks before printing entries

Gaslight v1 - Performance

- … not that great
- Issues:
  1. Tied to the number of cores on one system
  2. One plugin can require hundreds of thousands of runs
  3. Utilizing FUSE mounts is slow

Gaslight v2

- New work funded by NSF
- Performance:
  - Fully distributed
  - LD_PRELOAD to intercept reads
- Capabilities:
  - Finer grained intercepts
  - Beyond memory forensics tools

Conclusion

- We found programming errors in 14 Volatility plugins and 1 Rekall plugin through our automated fuzzing framework
- Performance currently slow, but v2 will see substantial improvements
- We would like to see all/more commonly used DFIR tools and libraries undergo fuzzing

Questions/Comments?

- ac@dfir.org / @attrc
- golden@cct.lsu.edu / @nolaforensix