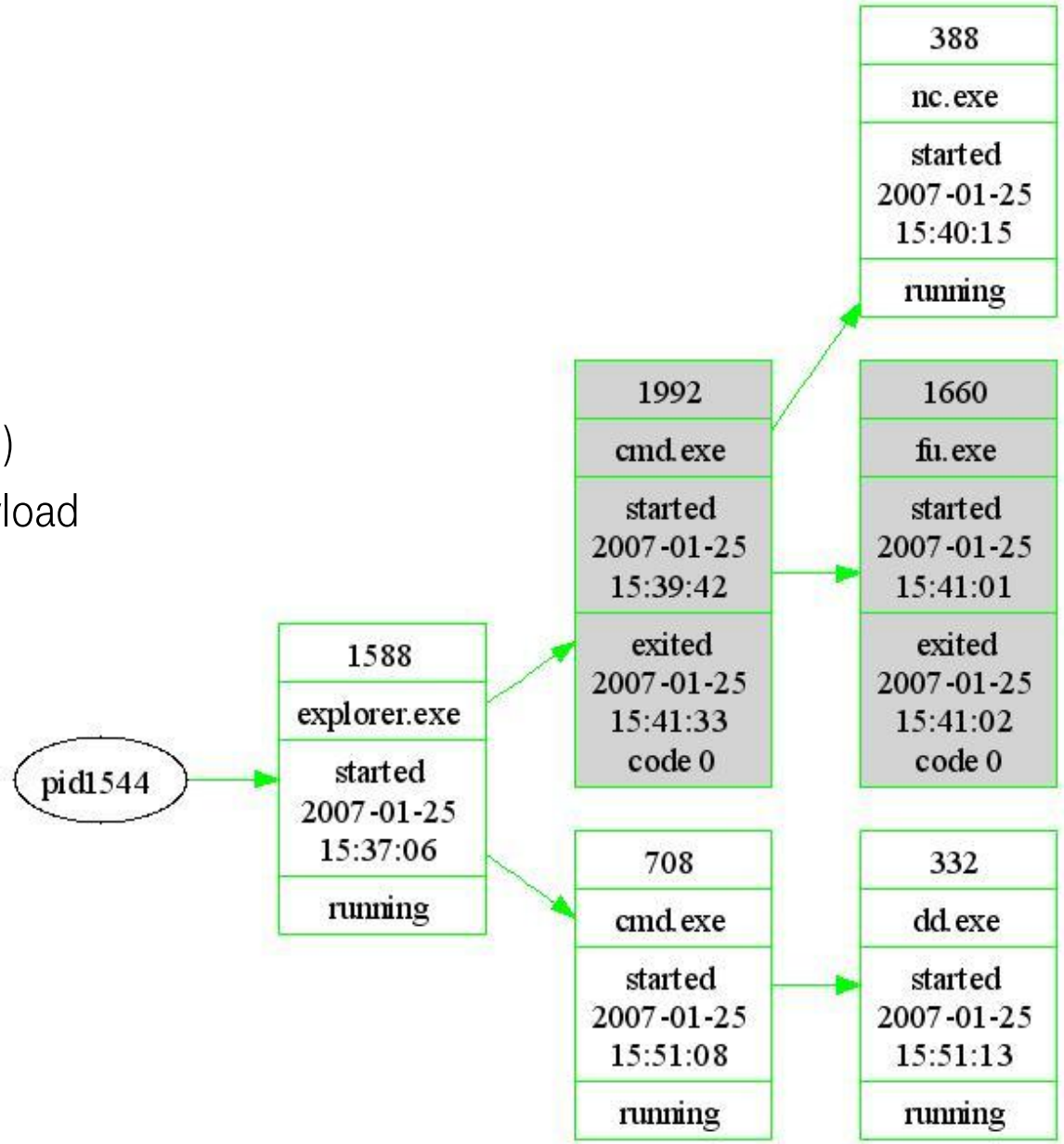
The impact of Microsoft Windows pool allocation strategies on memory forensics

Andreas Schuster

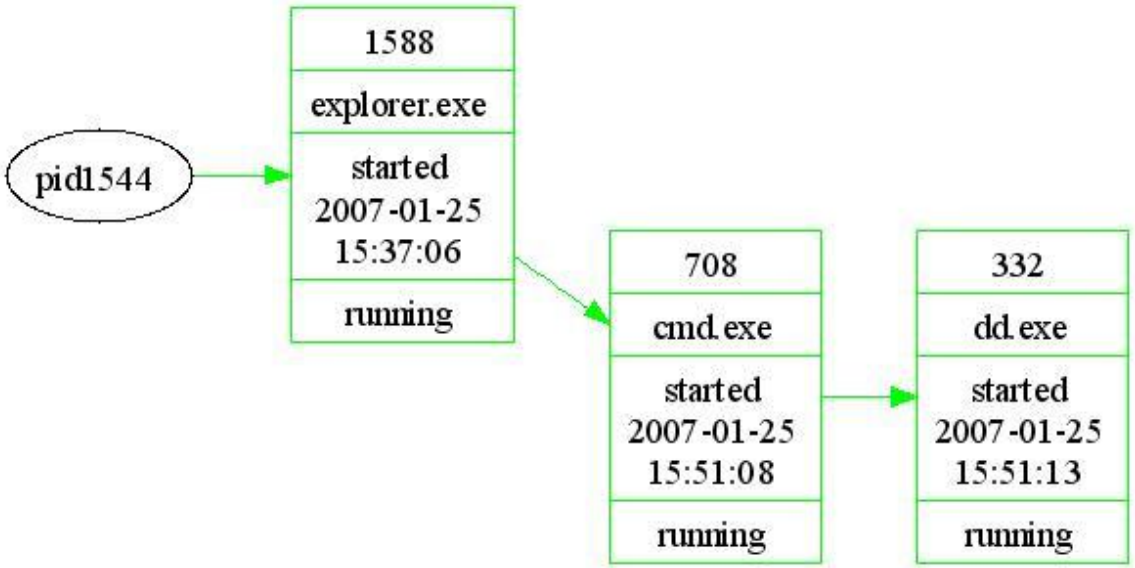
Introduction.
A simulated attack.

- Attacker
  - launches shell (cmd.exe)
  - launches “payload” (nc.exe)
  - launches fu.exe to hide payload
  - closes shell
- Incident Responder
  - launches (trusted) shell
  - obtains memory image



Introduction.
Expectation vs. Observation

- payload (netcat listener nc.exe) is visible, but “isolated”
- no evidence of terminated programs
  (attacker’s shell and rootkit)



Persistence of pool allocations.

Persistence of memory pool allocations. Related work.

- Farmer and Venema (2004) measured decay of freed memory on FreeBSD 4.1 and RedHat Linux 6.2. After “some ten minutes, about 90 percent of the monitored memory was changed“.
- Walters and Petroni (2007) counted changed memory pages on Windows XP SP2 running as VMware guest. After 15 hours of idle activity, 85% of 512 MB RAM were unchanged.
- Solomon, Huebner, Bem and Szeżynska (2007) used probe processes to measure the decay of userland data. “The majority of pages persisted for less than 5 min[utes] with single pages only lasting longer.”
- Chow, Pfaff, Garfinkel and Rosenblum (2005) filled network buffers in kernel space with marked and timestamped data. After 14 days of “everyday work” 3 MB out of initially 4 MB were still accessible.

Persistence of memory pool allocations. Test environment.

- Goal #1: avoid as much unwanted activity as possible

- deactivated unneeded system services
  - firewall,
  - background file transfer,
  - NTP client...

Persistence of memory pool allocations. Test environment.

- Goal #2: sampling shall not change the state of the observable

- run observed OS as guest in VMware
  - see Walters and Petroni, 2007
  - suspend VM to obtain the memory dump

Persistence of memory pool allocations.
Test environment.

- Goal #3: experiments shall be reproducible

- OS with prepared analysis environment (shell, debugger) stored as snapshot
- probe binaries and log files kept on host, accessed through VMware's shared folder
- test plan implemented as CMD batch

Persistence of memory pool allocations. Test plan.

1. launch probes no. 1 to 100
2. give the system time to settle down (5 minutes)
3. obtain memory image (reference) and scan for EPROCESS structures
4. terminate all probes
5. obtain memory image and scan for EPROCESS structures
6. repeat 1, 5, 15, 30, 60 minutes and 24 hours thereafter

Persistence of memory pool allocations. Results.

low file system activity:

- **90 EPROCESS structures after 24 hours**

- 8 ETHREAD, belonging to SYSTEM and svchost.exe
- 1 network related
- 1 not identified

high file system activity:

- **88 EPROCESS structures after 24 hours**

- 7 file system related data, e.g. MFT entries of probe files
- 3 ETHREAD belonging to background activity (svchost.exe, services.exe)
- 1 network related
- 1 VAD

Reuse of pool allocations.

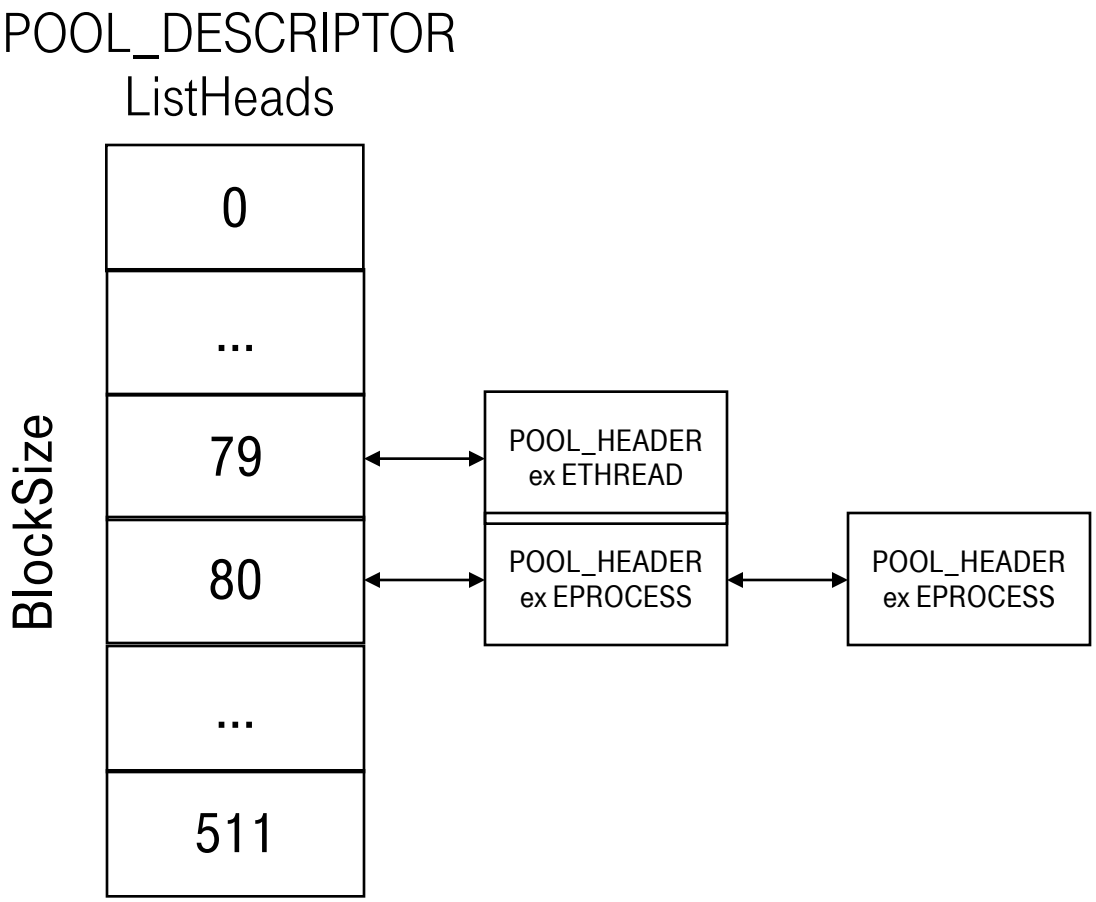
Reuse of pool allocations.
Related work.

- SoBelt (2005): How to exploit Windows kernel memory pool.
  http://xcon.xfocus.org/xcon2005/archives/2005/Xcon2005_SoBelt.pdf

- Johnson (2007): Memory Allocator Attack and Defense.
  http://seattle.toorcon.org/talks/richardjohnson.pptx

- Kortchinsky (2008): Real World Kernel Pool Exploitation.
  http://www.immunitysec.com/downloads/KernelPool.odp
  - detailed description of data structures and algorithms
  - offensive usage
  - highly recommended!

Reuse of pool allocations.
Keeping track of free allocations.



Reuse of pool allocations.
POOL_HEADER (allocated)

```
kd> dt _POOL_HEADER
nt!_POOL_HEADER
   +0x000 PreviousSize     : Pos 0, 9 Bits
   +0x000 PoolIndex        : Pos 9, 7 Bits
   +0x002 BlockSize        : Pos 0, 9 Bits
   +0x002 PoolType         : Pos 9, 7 Bits
   +0x004 PoolTag          : Uint4B
   +0x008 Payload
```

Reuse of pool allocations.
POOL_HEADER (free)

```
+0x000 PreviousSize      : Pos 0, 9 Bits
+0x000 PoolIndex         : Pos 9, 7 Bits
+0x002 BlockSize         : Pos 0, 9 Bits
+0x002 PoolType          : Pos 9, 7 Bits
+0x004 PoolTag           : Uint4B
+0x008 FreeList          : _LIST_ENTRY
   +0x000 Flink             : Ptr32
   +0x004 Blink             : Ptr32
+0x010 RemainingPayload
```

Reuse of pool allocations.
Test plan.

1. launch probes no. 1 to 3
2. terminate all probes in reverse order
3. obtain memory image and scan for EPROCESS structures
4. launch probe no. 4
5. obtain memory image and scan for EPROCESS structures

Reuse of pool allocations.
Results.

Probe no.	PID	EPROCESS	Page Directory Base Address
1	464	**0x04c9a020**	0x06bf1000
2	492	0x04878da0	0x01876000
3	500	0x01082da0	0x04b9f000
4	540	**0x04c9a020**	0x039f9000

Conclusion.

Conclusion.
Nonpaged pool.

- contains lots of meta-data about kernel objects and other objects (processes, threads, modules, files, network connections)
- no signs of active wiping and pool compaction
- data persists until
  - block of memory is reused
  - whole page is unused and gets removed from the pool

Conclusion.
Reuse of pool allocations.

- join adjacent free blocks

- reallocate:
  - matching size
    “EPROCESS overwrites EPROCES”
  - if there’s no free allocation of matching size, then use a larger one
    “ETHREAD overwrites EPROCESS”
  - prefer free allocations near the borders over those in the middle of the pool
    (buddy algorithm by Donald E. Knuth))

Conclusion.
Impact on memory acquisition tools.

- Installed prior to an incident (aka “Enterprise Forensic Solution”)
  - pre-allocate resources during initialization
  - activate resources when needed
- Installed post incident
  - use as little resources as possible
  - single thread
  - allow only 1 network connection
  - overlay instead of spawning a new process

Conclusion.
Measure the impact of IR/memory acquisition tools.

- Ian Sutherland, Jon Evans, Theodore Tryfonas, Andrew Blyth (2008): "Acquiring Volatile Operating System Data – Tools and Techniques"
  - memory footprint
    - page file bytes
    - virtual bytes
    - working set
  - time elapsed
  - impact on registry
  - use of DLLs

- proposal: also measure impact on pools, track calls to ExAllocatePoolWithTag

Conclusion.
Impact on memory analysis tools.

- first 8 bytes of payload overwritten, if allocation is marked as free (PoolType == 0)
- usually affects OBJECT_HEADER

- opportunity to improve signatures for pool allocations:
  - both pointers are pointing into kernel memory (upper half of address space)
  - alignment on 8-byte boundary
  - affects PoolFinder
  - identified more than 200 false-positives among 42.000 records

Questions?

Thank you for your attention!