Pool Allocations
in Windows Memory Forensics.

Andreas Schuster
Deutsche Telekom AG
Group Security
andreas.schuster@telekom.de

Pool Allocations in Windows Memory Forensics.
Agenda.

1. Introduction

2. Memory Management

3. Proof of Concept - PoolFinder
   3.1 Usage
   3.2 Example – Network Activity

4. Conclusion

5. Questions & Answers

Introduction.
Assumptions.

- Intel 32bit architecture.
- Microsoft NT family / Vista kernel.
- less than 4 GiB of main memory.
- virtual memory evenly split among kernel and userland, that is no /3GB switch.

Introduction.
Why Is There a Need for Memory Forensics?

„Pulling the Plug“ is still a common procedure.

But ...

- certain attacks don‘t leave traces on disk
- a system‘s state is kept in its memory only
  - processes / threads
  - driver code
  - interfaces, listening sockets, TCP connections
  - ... and much more

Introduction.
Memory Management.

Most of this state information is kept in kernel memory.

Kernel memory is a rare resource, so data structures tend to be small.

Memory is managed in units called „pages“.

Page sizes supported by the Intel Pentium MMU:

- 1 kiB not used
- 4 kiB common size
- 4 MiB HAL/kernel image

A finer-grained memory management is needed.

Windows Memory Management.
Memory Pools.

Solution:

- Several memory pages are combined to form a „pool“.
- Requests smaller than page size are served from the pool.
- Concept also known as „heap“.

Different pools:

- non-paged pool – always resides in memory
- paged pool – may be paged-out to disk

Windows Memory Management.
API.

Set of Allocators:

- nt!ExAllocatePool - deprecated
- **nt!ExAllocatePoolWithTag** – common form
- nt!ExAllocatePoolWithQuotaTag – charges current process
- nt!ExAllocatePoolWithTagPriority – specifies importance of request
- ...

Matching set of Deallocators:

- nt!ExFreePool
- **nt!ExFreePoolWithTag**
- ...

Some subsystems provide their own set of (de)allocators.

- FsRtlAllocatePoolWithTag

Windows Memory Management.
_POOL_HEADER Overview.

_POOL_HEADER describes not the whole pool, but a single pool allocation. The structure is semi-documented; information can be retrieved from debug symbols.

```
>dt 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
   +0x004 AllocatorBackTraceIndex : Uint2B
   +0x006 PoolTagHash               : Uint2B
```

Windows Memory Management.
BlockSize and PreviousSize.

BlockSize:

- size of this allocation
- pointer to next allocation

PreviousSize:

- size of the previous allocation
- pointer to previous allocation
- 0 for the first allocation in a page

Both:

- measured in units of 8 bytes (Windows 2000: 32 bytes).
- includes the _POOL_HEADER (8 bytes), so must be 1 at least

Windows Memory Management.
PoolType (Programmer's View).

Declared in Windows Development Kit, file wdm.h:

```
typedef enum _POOL_TYPE {
  NonPagedPool,                                    0000
  PagedPool,                                       0001
  NonPagedPoolMustSucceed,                         0010
  DontUseThisType,                                 0011
  NonPagedPoolCacheAligned,                        0100
  PagedPoolCacheAligned,                           0101
  NonPagedPoolCacheAlignedMustS,                   0110
  MaxPoolType                                      0111
} POOL_TYPE;
```

Values greater than 31 exist and indicate a session pool.

Windows Memory Management.
PoolType (In-Memory View).

Pool type increased by 1.

Distinction:

- 0 = block is free (deallocated)
- odd = non-paged pool
- even = paged pool

Windows Memory Management.
PoolTag.

According to documentation of ExAllocatePoolWithTag in MSDN:

- up to 4 character literals
- ASCII values between 0 and 127
- stored in little-endian (reverse) byte-order
  '1234' stored as '4321'
- every allocation code path should use a unique pool tag

There is no registry for pool tags.
Every application is free to use any pool tag!

Accidental and intentional misuse of well-known pool tags
(see skape & Skywing 2005 on Microsoft PatchGuard x64)

PoolFinder.
About the Tool.

Proof of concept:

- implements a set of rules (see proceedings for details)
- flexible, written in Perl
- works on (almost) any dump file format
  - raw dumps (dd, WinHex Capture, KnTTools)
  - VMware 4.x/5.x suspended sessions
  - crash dumps (DMP, kernel and full format)
  - page files (pagefile.sys)

http://computer.forensikblog.de/files/poolfinder/

PoolFinder.
Usage.

poolfinder.pl [options] *file*

Some important options:

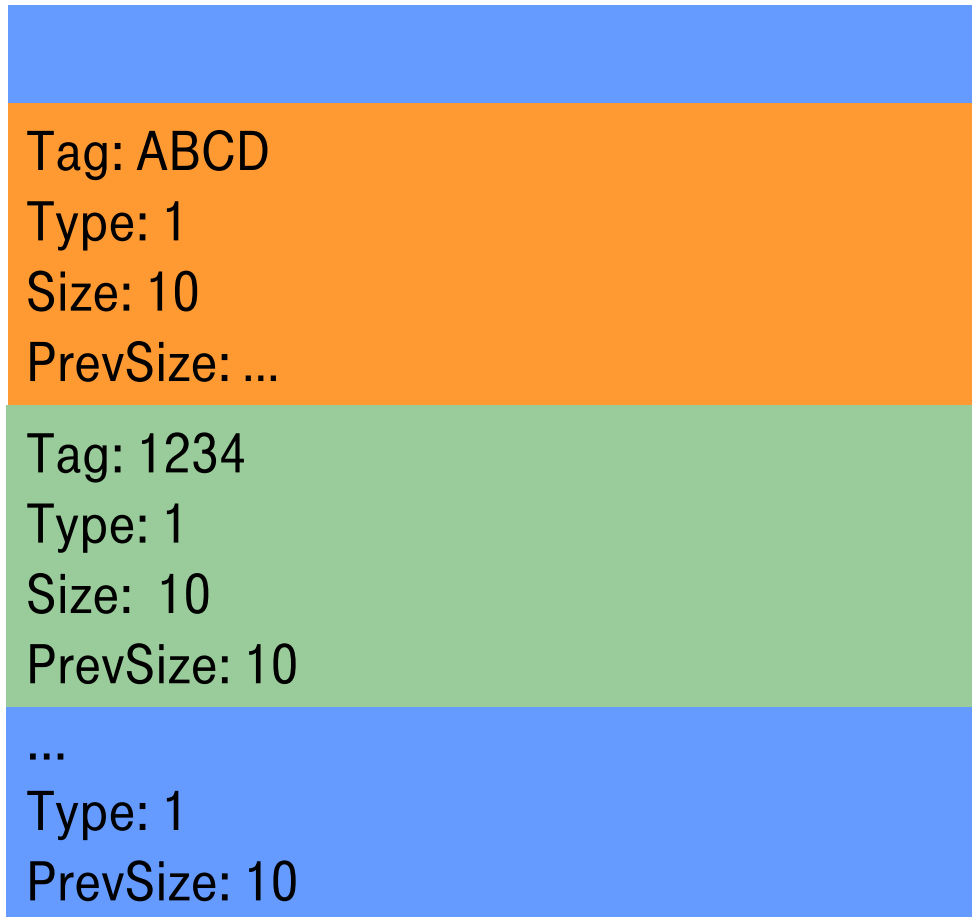
- `--help` - read it, please!
- `--win2000` - enables support for Windows 2000
- `--level` - adjusts required score
- `--pagefile` - enables filtering suitable for page files
- `--strictags` - allows only printable characters in pool tags

PoolFinder.
Sample Output.

No.	Tag	EPROCESS	Size	Offset	P	F	Type	Indx
1	Thre		632	0x0004e000	P	-	0x05	0x00
2		0xe152fbf0	16	0x0004e278	-	F	0x00	0x00
3	Sema		56	0x0004e288	P	-	0x05	0x00
4	Vadl		24	0x0004e2c0	-	F	0x00	0x00
5	Ntfn		40	0x0004e2d8	-	-	0x05	0x00
6	Ntfr		64	0x0004e300	-	-	0x05	0x00
7	LBtn		72	0x0004e340	-	-	0x05	0x00
8	Even		8	0x0004e388	P	F	0x00	0x00
1584	Wait		1112	0x0056a690	-	F	0x00	0x00
1585	Wait		1088	0x0056a6a8	-	f	0x01	0x00
70338	Gh05		1160	0x07f66b78	-	-	0x06	0x02

PoolFinder.
Memory Deallocation (1).

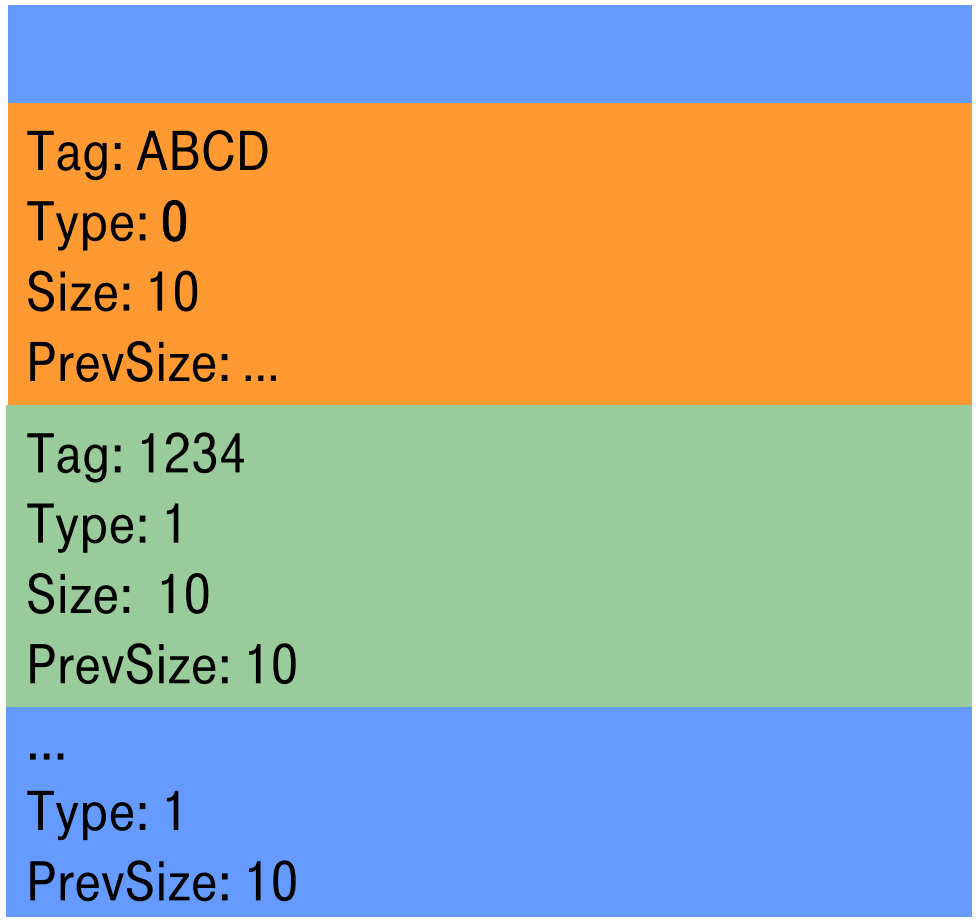
Two allocations tagged „ABCD“ and „1234“ in a pool.



PoolFinder.
Memory Deallocation (2).

The first allocation is freed.

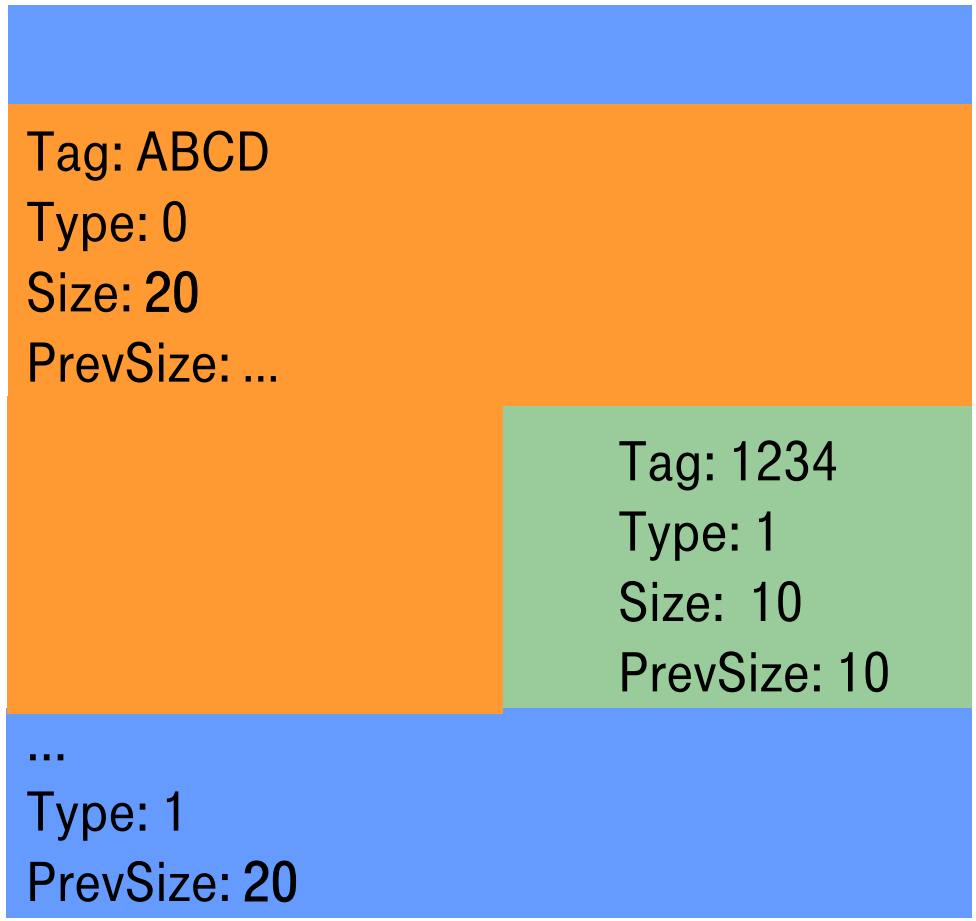
Notice the changing „Type“.



PoolFinder.
Memory Deallocation (3).

Now the second allocation is freed, too.

The second allocation is not marked as free. Instead it is merged into the first one.



PoolFinder.
Memory Deallocation (4).

Left side:
A debugger only sees allocated and (explicit) free blocks.

Right side:
PoolFinder in addition sees merged (implicit free) blocks.

Example – Network Activity.
The Situation.

Scenario:

- Microsoft Windows XP SP 2 running as guest OS in VMware 5.5.x
- Netcat (nc.exe) is started, listening on port 666/tcp.
- Some minutes later Netcat is shut down.
- There was no incoming connection,
  so there‘s no suspicious network send/receive buffer.
- VMware session is suspended to „dump the physical memory“.

Goal: Find any evidence that Netcat was listening on port 666/tcp.

Example – Network Activity.
Step 1 – Find the Proper Routine and Code Path.

What we're looking for is related to TCP/IP networking.

Start at %WINDIR%\System32\drivers\tcpip.sys

tcpip!TdiOpenAddress

...

```
.text:0001CD5D          push    NormalPagePriority
.text:0001CD5F          push    'APCT'          ; Tag
.text:0001CD64          push    360             ; NumberOfBytes
.text:0001CD69          push    NonPagedPool
.text:0001CD6B          call    ds:__imp__ExAllocatePoolWithTagPriority@16
```

...

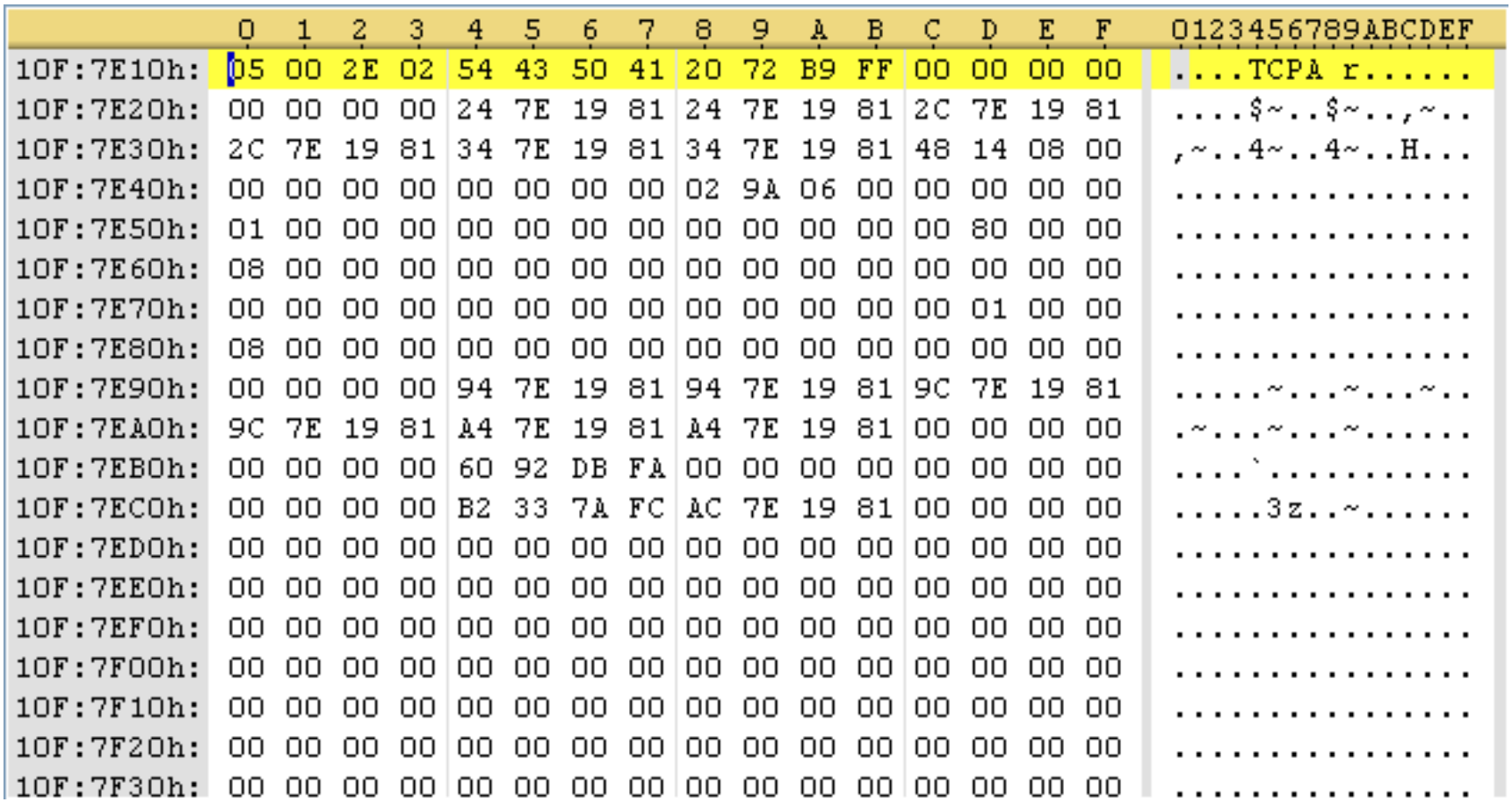
Example – Network Activity.
Step 2 – Create a Signature.

PoolFinder identifies about 51,000 allocations.

What are we interested in?

- Tag: TCPA
  16 allocations left
- Size: 360+8
  14 allocations left
- Type: odd (non-paged pool) or zero (free)
  14 allocations left

Example – Network Activity.
Preliminary Results.



	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0123456789ABCDEF
1OF:7E10h:	05	00	2E	02	54	43	50	41	20	72	B9	FF	00	00	00	00	....TCPA r.....
1OF:7E20h:	00	00	00	00	24	7E	19	81	24	7E	19	81	2C	7E	19	81	....$~..$~..,~..
1OF:7E30h:	2C	7E	19	81	34	7E	19	81	34	7E	19	81	48	14	08	00	,~..4~..4~..H...
1OF:7E40h:	00	00	00	00	00	00	00	00	02	9A	06	00	00	00	00	00	....
1OF:7E50h:	01	00	00	00	00	00	00	00	00	00	00	00	00	80	00	00	....
1OF:7E60h:	08	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
1OF:7E70h:	00	00	00	00	00	00	00	00	00	00	00	00	00	01	00	00	....
1OF:7E80h:	08	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
1OF:7E90h:	00	00	00	00	94	7E	19	81	94	7E	19	81	9C	7E	19	81	.....~...~...~..
1OF:7EA0h:	9C	7E	19	81	A4	7E	19	81	A4	7E	19	81	00	00	00	00	.~...~...~......
1OF:7EB0h:	00	00	00	00	60	92	DB	FA	00	00	00	00	00	00	00	00	....`...........
1OF:7EC0h:	00	00	00	00	B2	33	7A	FC	AC	7E	19	81	00	00	00	00	.....3z..~......
1OF:7ED0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
1OF:7EE0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
1OF:7EF0h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
1OF:7F00h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
1OF:7F10h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
1OF:7F20h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....
1OF:7F30h:	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	....

Example – Network Activity.
Step 3 – Learn to Interpret the Data.

```
.text:0001CF4D          mov    eax, [ebp+var_LocalAddress]
.text:0001CF50          mov    [esi+44], eax
.text:0001CF53          mov    al, byte ptr [ebp+arg_Protocol]
.text:0001CF56          mov    [esi+50], al
...
.text:0001CF5C          mov    [esi+48], di    ; LocalPort
...
.text:0001CF76          call   _PsGetCurrentProcessId@0
.text:0001CF7B          mov    [esi+328], eax
.text:0001CF81          lea    eax, [esi+344]
.text:0001CF87          push   eax             ; CurrentTime
.text:0001CF88          call   ds:__imp__KeQuerySystemTime@4
```

Example – Network Activity.
Step 4 – The Results.

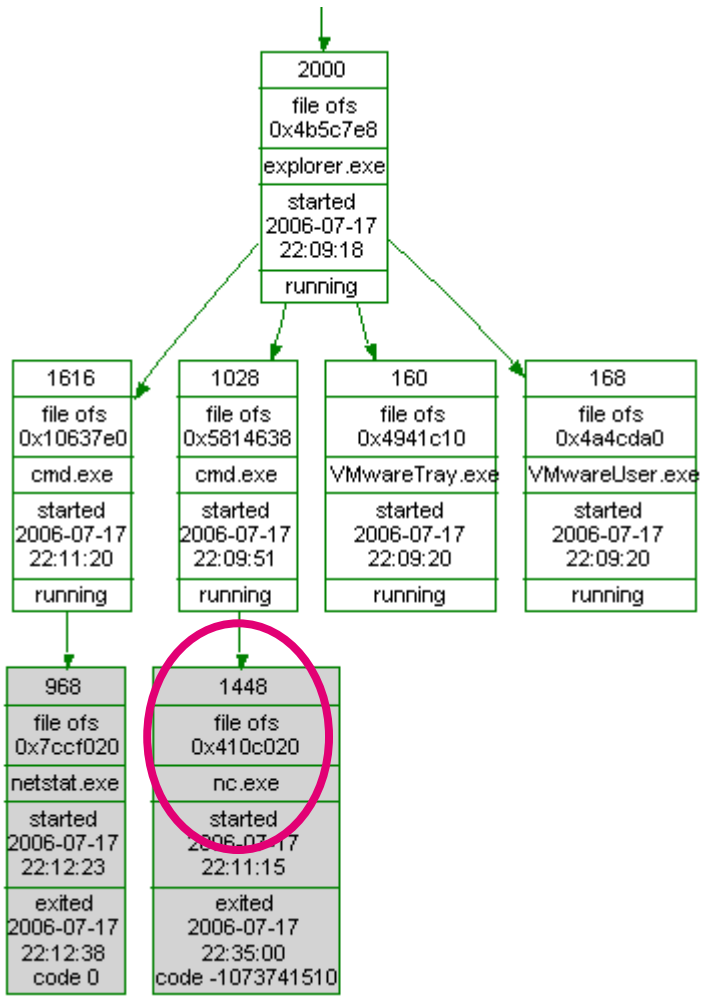
1. 192.168.186.128:138/UDP, PID=4, 2006-07-17 22:08:47
2. 0.0.0.0:135/TCP, PID=800, 2006-07-17 22:08:40
3. 0.0.0.0:0/IGMP, PID=884, 2006-07-17 22:08:49
4. 0.0.0.0:0/GRE, PID=4, 2006-07-17 22:08:51
5. 0.0.0.0:1029/UDP, PID=948, 2006-07-17 22:09:46
6. 127.0.0.1:1025/TCP, PID=1508, 2006-07-17 22:08:51
7. **0.0.0.0:666/TCP, PID=1448, 2006-07-17 22:11:15 (defunct)**
8. 192.168.186.128:139/TCP, PID=4, 2006-07-17 22:08:47

...

Example – Network Activity.
Step 4 – The Results.

A search for (hidden/terminated) processes by PTFinder produces the following process tree.

Note the matching PID: 1448



Example – Network Activity.
TCP Connection Objects.

One can repeat the process for TCP connection objects. This results in:

192.168.186.128:1037 -> 213.253.9.70:80, PID=884
192.168.186.128:1038 -> 213.253.9.70:80, PID=884
192.168.186.128:1039 -> 64.4.21.93:80, PID=884

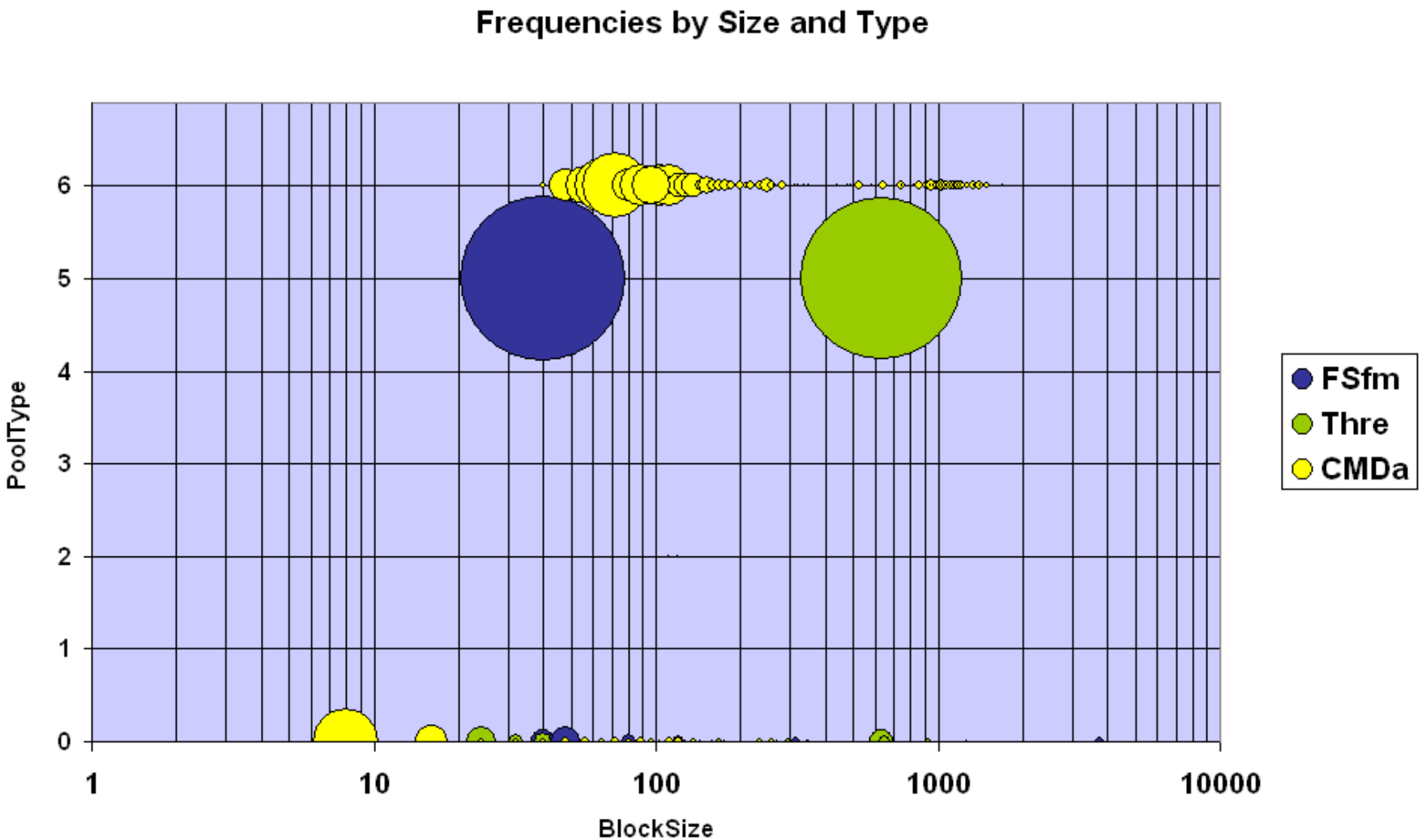
PID 884 identifies svchost.exe.

Remote IP addresses belong to Microsoft and a content distribution network.

Conclusion.
First Results.

- helps to explore memory pools and page files
- shows deleted data
- generates a significant amount of false-negatives / false–positives, depending on options
- default options avoid clutter at the cost of false-negatives
- does not recognize „untagged“ pool allocations

Conclusion.
Further Work - Baselining (1).



Conclusion.
Further Work - Baselining (2).

Other aspects to look at:

- entropy by PoolTag (and BlockSize, PoolType)
- usage of PoolTags by certain OS versions

  e.g. Atom (Windows 2000) vs. AtmA and AtmT (XP)

Conclusion.
Further Work - Persistance of Data.

Chow, Garfinkel, Tal and Rosenblum (2005) found marker sequences in network buffers 14 days after injection.

Redo the experiment and place data in tagged pool allocations. Measure decay rate over time, depending on

- operating system
- pool class (paged, non-paged)
- allocation size
- load profile (desktop, web server, database)

This could help in optimizing IR procedures (cost/speed trade-off).

Questions and Answers.

Andreas Schuster
Deutsche Telekom AG
Konzernsicherheit
andreas.schuster@telekom.de

Thank You for Your Attention.

Andreas Schuster
Deutsche Telekom AG
Konzernsicherheit
andreas.schuster@telekom.de