Adobe Reader's
Custom Memory Management:
a Heap of Trouble

--

Guillaume Lovet, Threat Response, Sr. Manager
Hafei Li, Sr. Security Researcher

Objectives

- Gain detailed knowledge on **Adobe Reader**'s **Custom Heap Management** System

- Become aware of the security **issues** it raises

  (the bad guys know, you must know too)

- Be given insights on how to **leverage** them, in the frame of an exploitation scenario

  (useful for penetration testing, mitigation research, threat response...)

Introduction

- 80% of exploits in the Wild in Q4 2009 were PDF ones
  => **1st choice exploitation vector**

- Why?
  - Ubiquity of Adobe Reader
  - Widespread false beliefs about viruses
  - Patching process not integrated in Win. Updates
  - Complexity of the specifications...

- Late 2009: new "high-risk PDF 0-day vuln exploited in the Wild" (CVE-2009-3459)

- Analysis revealed interesting techniques -- we digged deeper

Custom Heap Management on Adobe Reader

- Traditional programs outsource memory storage to the OS (via system calls)
- For performance reasons, Adobe Reader implements its own, on top of the OS
- Resembles a Cache
- One top level structure: **Acro Block**
- Two underlying structures/systems:
  - **Acro Cache Block**
  - **BIB Block**

Agenda

1	Acro Blocks
2	The Acro Cache
3	Exploiting the Acro Cache
4	The BIB Cache
5	Exploiting the BIB Cache

Acro Blocks - in Memory

```
+---------------+
|               |
|  Acro Header  |
|  (24 bytes)   |
|               |
+---------------+
| lpAcroHeader  |
|   (4 bytes)   |
+---------------+
|               |
|               |
|  Block Data   |
| (dwDataSize)  |
|               |
|               |
+---------------+
```

Acro Blocks - Data Structures

```
struct acro_header
{
    acro_managing_pool* lpAcroPool;      // Points to the Acro Managing Pool
    DWORD               reserved         // Set to 0
    DWORD               flag;            // Type of Block. Set to 2 for Acro Blocks
    acro_header*        Blink;           // Previous Acro Header in the list
    acro_header*        Flink;           // Next Acro Header in the list
    DWORD               dwDataSize       // Size of the Block Data
}


struct acro_managing_pool
{
    DWORD           reserved[3];
    cache_managing* lpCacheManaging[32]; // Managing structures for the Acro Cache
    DWORD           reserved;
    acro_header*    lp_head_acro_header; // Header of the first Acro Block in the list
}
```

Acro Blocks - Organization

```
                             +------------------------------------------+
                             |    ------------------+                   |
                             v    v                 |                   |
+------------+    +--------------------+    +--------------+    +--------------+
| 0x014dce40 |--->| Acro Managing Pool |--->| Acro Block 1 |--->| Acro Block 2 |--->  ...
+------------+    +--------------------+    |              |<---|              |<---
                                            +--------------+    +--------------+
```

Agenda

1	Acro Blocks
2	The Acro Cache
3	Exploiting the Acro Cache
4	The BIB Cache
5	Exploiting the BIB Cache

Acro Cache - in Memory

```
+--------------------+
|                    |
|     Acro Header    |
|     (24 bytes)     |
|                    |
+--------------------+
|     lpAcroHeader   |
+------------------- +
|                    |
|    Cache Header    |
|     (24 bytes)     |
|                    |
+--------------------+
|    lpCacheHeader   |
+--------------------+
|     Cache Block 1  |
|    (dwBlockSize)   |
+--------------------+
         ...
+--------------------+
|    lpCacheHeader   |
+--------------------+
|   Cache Block 128  |
|    (dwBlockSize)   |
+--------------------+
```

Acro Cache - Data Structures

```
struct cache_header
{
    cache_managing* lpCacheManaging;  // Pointer to the Cache Manager structure
    DWORD           dwAllocatedBlocks;// Number of allocated Cache Blocks in this Acro Cache
    DWORD           flag;             // Type of object. Set to 0 for Acro Cache
    cache_header*   Blink;            // Previous Acro Cache
    cache_header*   Flink;            // Next Acro Cache
    DWORD           dwBlockSize;      // Size of contained Cache Blocks
}

struct cache_managing
{
    acro_managing_pool* lpAcroPool;          // Pointer to the Acro Managing Pool
    free_cache_block*   lp_head_free_block;  // Head of Free Cache Blocks list
    cache_header*       lp_head_cache_header; // Head of Acro Caches list
    DWORD               dwBlockSize;         // Managed Cache Blocks size
}


struct free_cache_block
{
    free_cache_block* Blink;     // Previous free cache block
    free_cache_block* Flink;     // Next free cache block
}
```

Acro Cache - Organization

```
                                                     +--------------------+     +--------------------+
                                                     |    Acro Header     |     |    Acro Header     |
                                                     |                    |     |                    |
                                                     +--------------------+     +--------------------+
                                                     |    lpAcroHeader    |     |    lpAcroHeader    |
+---------------+      +--------------------+        +--------------------+     +--------------------+
| Acro Managing |      | Acro Cache Managing |       |                    |---->|                    |---->
|     Pool      |---+--->|                  |---->|    Cache Header    |     |    Cache Header    |      ...
|               |   |  |  (dwBlockSize = 8) |       |                    |<----|                    |<----
+---------------+   |  +--------------------+        +--------------------+     +--------------------+
                    |                                |                    |     |                    |
                    |                                |                    |     |                    |
                    |                                | 128 lpCacheHeaders |     | 128 lpCacheHeaders |
                    |                                |         &          |     |         &          |
                    |                                |  128 Cache Blocks  |     |  128 Cache Blocks  |
                    |                                |                    |     |                    |
                    |                                |                    |     |                    |
                    |                                +--------------------+     +--------------------+
                    |
                    |  +--------------------+
                    |  | Acro Cache Managing |
                    +--->|                  |----> ...
                    |  |  (dwBlockSize = 12) |
                    |  +--------------------+
                    .
                    .          ...
                    .
                    |  +--------------------+
                    |  | Acro Cache Managing |
                    +--->|                  |----> ...
                       | (dwBlockSize = 128) |
                       +--------------------+
```

Acro Cache - Zoom on Free Blocks

```
                                                      Acro Cache              Acro Cache
                                                    +--------------+        +--------------+
                                             +----->| Cache Header |<-------| Cache Header |<----------- ...
                                             |      |              |------->|              |----------->
+---------------------+ lp_head_cache_header |      +--------------+        +--------------+
| Acro Cache Managing |----------------------+      |              |        |              |
|                     |                             .              .        .              .
|                     | lp_head_free_block          .              .        .              .
|  (dwBlockSize = 8)  |----------------------+      |              |        |              |
+---------------------+                      |      +--------------+        +--------------+
                                             |      | lpCacheHeader|        | lpCacheHeader|
                                             |      +--------------+        +--------------+
                                             +----->|  Free Block  |<-------|  Free Block  |<------+
                                                    |  (8 bytes)   |------->|  (8 bytes)   |----+  |
                                                    +--------------+        +--------------+    |  |
                                                    |              |        |              |    |  |
                                                    .              .        .              .    |  |
                                                    .              .    +-----------------------+  |
                                                    .              .    |   .              .       |
                                                                        |   |              |       |
                                                                        |   +--------------+       |
                                                                        |   | lpCacheHeader|       |
                                                                        |   +--------------+       |
                                                                        +-->|  Free Block  |-------+
                                                                            |  (8 bytes)   |-----------> ...
                                                                            +--------------+
                                                                            |              |
```

Acro Cache - Allocation

- Acro Cache system = acro_allocate()
- Used by basic functions (eg: stream decoding, processing top objects in PDF such as "/Pages", "/Page", etc...)
- General logic:
  - **Requested Size > 128 bytes**
    - allocates a "direct" Acro Block (asking the OS for heap space)
    - Returns pointer to its data block
  - **Requested Size <= 128 bytes**
    - Looks for an appropriate Free Cache Block
    - Unlinks it rom the Free Cache Blocks list
    - Returns a pointer to it

Acro Cache - Initialization

```
                           Cache Block            Cache Block                  Cache Block

+-------------+         +---------------+      +---------------+           +---------------+
| Acro Cache |  NULL <----|     Blink     |<----|     Blink     |<---- ... <----|     Blink     |
|  Managing  |          +---------------+      +---------------+           +---------------+
|            |----------->|     Flink     |---->|     Flink     |----> ... ---->|     Flink     |----> NULL
+-------------+         +---------------+      +---------------+           +---------------+
                        |               |      |               |           |               |
                        | Uninitialized |      | Uninitialized |           | Uninitialized |
                        |               |      |               |           |               |
                        +---------------+      +---------------+           +---------------+

                        <--------------------------- Memory grows this way

                         Allocation is done this way ----------------------->
```

Acro Cache - Unallocation

General Logic of acro_free():

1. locates the header (with lpHeader or lpCacheHeader)

2. Identifies the type of block

3. If Cache Block
- adds it to the head of the Free Cache Block list of its kind

4. If Acro Block
- unlinks it from the Acro Block list
- Asks the OS to free it

Agenda

1	Acro Blocks
2	The Acro Cache
3	Exploiting the Acro Cache
4	The BIB Cache
5	Exploiting the BIB Cache

Strategies

- Two main ways to exploit Heap corruption flaws:
    - **Overwrite some application-provided** data in the Heap
    - **Corrupt the internal structures** used by the Heap management (block headers, etc...)
- Today, limited efficiency with OS Heap management systems:
    - "safe unlinking" since SP2
    - Heap state hard to predict across executions
- In Acro Cache case, both strategies are relevant

Overwriting App Data

- Assuming a vulnerable Acro Cache Block, 2 essential questions:
  - *Is there data within a Cache Block in the same Acro Cache that pertains to the execution flow?*
  - *Is the distance between this targeted Cache Block and our vulnerable Block predictable enough?*

- The Key Pointer
  - v-pointer => points to fixed address (the v-table)
  - Frequent on the Heap

- Predictability
  - Opening a basic document several times in a row => Cache for big blocks are the most stable
  - Let's use biggest (128 bytes) for experiment

Overwriting App Data (II)

0:007> dd poi(poi(poi(0x014D71E8) + 0x0C + 31*4) + 4)

```
0200bc14  00000000 0200bb90 89037a1b 1b476493
0200bc24  00030007 00000000 00000000 00000000
0200bc34  00000000 00000000 00000000 00000000
0200bc44  00000000 00000000 00000000 00000000
0200bc54  00000000 00000000 00000000 00000000
0200bc64  00000000 00000000 00000000 00000000
0200bc74  00000000 00000000 00000000 00000000
0200bc84  00000000 00000000 00000000 00000000
```

0:007> dd poi(poi(poi(0x014D71E8) + 0x0C + 31*4) + 4) + 132 + 132

Overwriting App Data (III)

```
0:007> dd poi(poi(poi(0x014D71E8) + 0x0C + 31*4) + 4 )

0200bc14  44444444 44444444 44444444 44444444
0200bc24  44444444 44444444 44444444 44444444
...
0200bd1c  44444444 44444444 55555555
```

- Then resume execution -

```
(380.298): Access violation - code c0000005 (first chance)

009d993f 833858          cmp     dword ptr
[eax],58h  ds:0023:55555555=????????

0:000> u eip
```

Corrupting the structures

When an Acro Block is unlinked:

lpAcroHeader->Flink->Blink = lpAcroHeader->Blink;
lpAcroHeader->Blink->Flink = lpAcroHeader->Flink;

Translates to:

[[lpAcroHeader + 0x10] + 0x0C] = [lpAcroHeader + 0x0C]
[[lpAcroHeader + 0x0C] + 0x10] = [lpAcroHeader + 0x10]

Corrupting the structures (II)

- In an exploitation scenario: overwrite lpAcroHeader (or lpCacheHeader) => points to a forged header:

```
+----------+
| AAAAAAAA | <- lpAcroPool
+----------+
| BBBBBBBB | <- reserved DWORD
+----------+
| CCCCCCCC | <- Type flag
+----------+
| DDDDDDDD | <- Blink
+----------+
| EEEEEEEE | <- Flink
+----------+
| FFFFFFFF | <- dwDataSize
+----------+
```

When Unlinked:

[EEEEEEEE + 0x0C] = DDDDDDDD
[DDDDDDDD + 0x10] = EEEEEEEE
This is equivalent to:

[X] = Y
[Y + 0x10] = X - 0x0C

Agenda

1	Acro Blocks
2	The Acro Cache
3	Exploiting the Acro Cache
4	The BIB Cache
5	Exploiting the BIB Cache

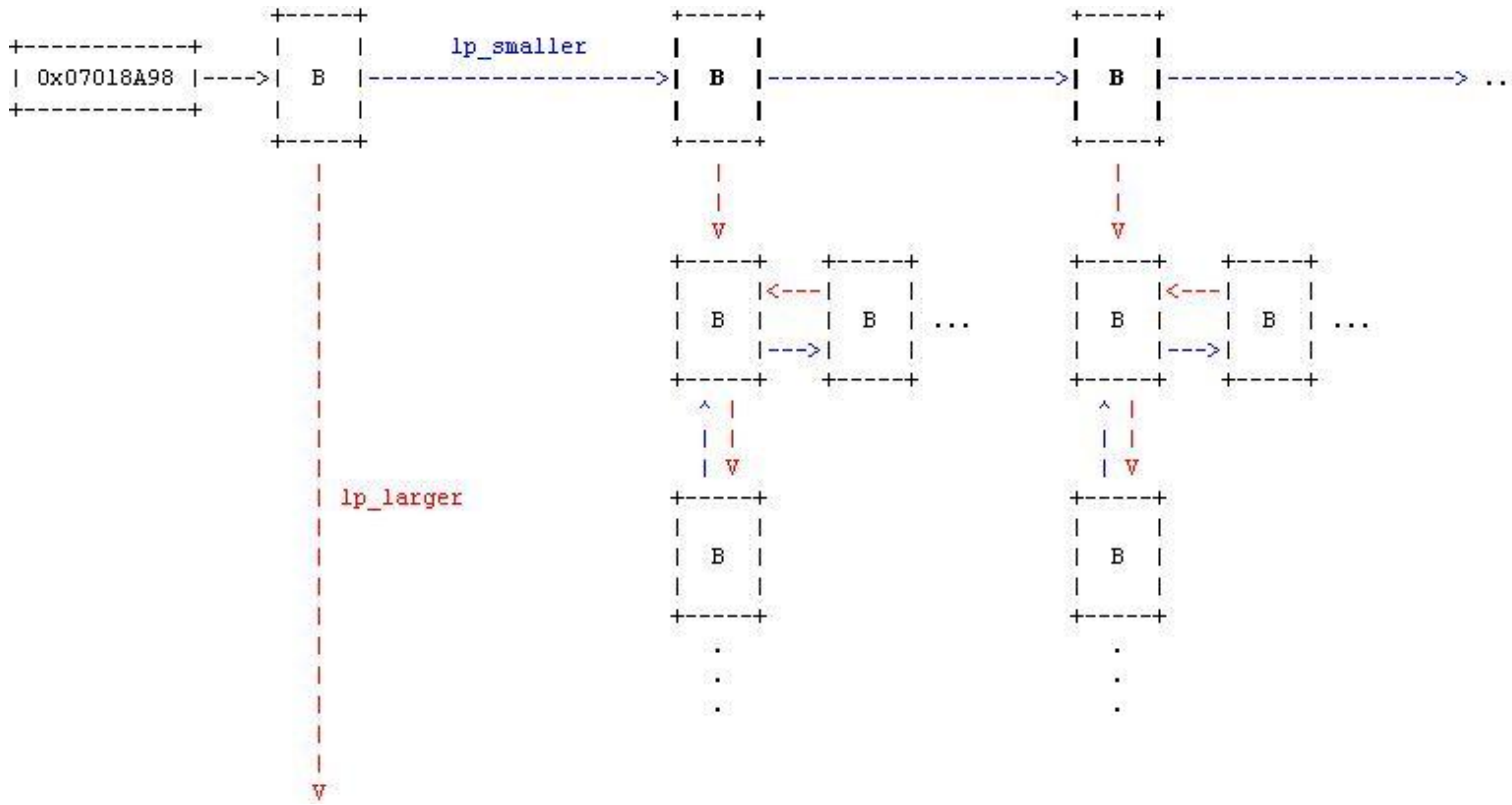
BIB Cache - In Memory

```
+-------------------+
|    Acro Header    |
|      (0x18)       |
|                   |
+-------------------+
|   lpAcroHeader    |
|      (0x04)       |
+-------------------+
|     Reserved      |
|      (0x04)       |
+-------------------+
|   flag_s (0x02)   | <-- 0x0000
+-------------------+                                           ---+
|   b_size (0x02)   | <-- Size of the following BIB Block          |
+-------------------+                                              |
|    BIB Block 1    |                                              |
|    (b_size1)      | <-- BIB Block                                +-- One BIB unit
|                   |                                              |
+-------------------+                                              |
|  b_size_f (0x02)  | <-- Size of the previous BIB Block           |
+-------------------+   & Status Flag for the next BIB Block    ---+
|   b_size (0x02)   |
+-------------------+
|    BIB Block 2    |
|    (b_size2)      |
|                   |
+-------------------+
|  b_size_f (0x02)  |
+-------------------+
         ...
+-------------------+
|   flag_e (0x02)   |
+-------------------+
```

BIB Cache - Free BIB Blocks

```
struct free_bib_block
{
     free_bib_block*   lp_smaller;          //  Pointer  to  a  smaller  free  bib
block
    free_bib_block*  lp_larger;          // Pointer to a larger free bib block
    DWORD                reserved;
    DWORD                reserved;
     free_bib_block*   lp_pre_same_size;  //  Blink  to  a  free  bib  block  of  same
size
     free_bib_block*   lp_next_same_size; //  Flink  to  a  free  bib  block  of  same
size
}
```

BIB Cache - Organization



BIB Cache - Allocation

high-level logic of *bib_allocate*:

- If requested size > than 65024 (0xFE00) bytes, a classical Acro Block allocated and returned
- Pulls *the smallest node whose size is bigger than the requested size* (if more than one, the first same-size)
- If that node is bigger than the requested size by an amount of 28 bytes (0x1C), the node is divided in two:
    - first part (of requested size) returned to the requester
    - second part inserted in the cache at the appropriate place (unique)
- Otherwise, the whole node is returned to the requester for memory storage.

Agenda

1	Acro Blocks
2	The Acro Cache
3	Exploiting the Acro Cache
4	The BIB Cache
5	Exploiting the BIB Cache

Corrupting the Structures

- Overwriting lpAcroHeader
  - Works in Underflow Cases
  - Branches to the case described earlier (unlink attack)
  - Useless in overflow, use-after-free, etc...

- Overwriting lp_next_same_size
  - Points to a forged Free BIB Block
  - But what to do with it??
  - Let's have a closer look at the insertion procedure (for new free BIB blocks)

Insertion procedure

```
DWORD block_size = (DWORD)*(USHORT *)(lpBibBlock - 2);

//if the bib block size is 0xFE01, handle it as an acro block
if ((block_size == 0xFE01) && (lpBibBlock != NULL))
{
  //locate the acro block pointer
  unsigned char *lpAcroBlock = lpBibBlock - 8;
  //obtain the value of "reserve"
  v_reserve = *(DWORD *)(lpBibBlock - 8);
  if (v_reserve >= 0x00020000)
  {
    //free the acro block
    acro_free(lpAcroBlock);
  }
}
```

Corrupting the Structures (II)

- If the free block to insert has a size of 0xFE01 bytes => occupies a full Acro Block, which is thus freed!
- Upon allocation, a large enough Free BIB Block is divided in 2...
- ... And the reminder new BIB Block is inserted in the Cache
- Thus, we craft our forged BIB Block so that the reminder is 0xFE01 bytes => the insertion procedure will attempt to free its container Acro Block
- This means unlinking it... Game Over

Demo

Conclusion

- Custom Heap Management may be faster, but lacks all the security mechanisms OS has
- Empowers attackers with the capacity to exploit Heap Corruption vulnerabilities (once were hard to leverage)
- In a context where PDFs are a prime infection vector (eg: Ghostnet) for targeted attacks, must be addressed
- Good news: has already been, at the OS level (safe unlinking, heap metadata cookies, etc...)

Thank You