Images, iPhone, bugs & NSO

Whoami

@garfie_d

Ex Abertay 2015-2020

Ex Royal Holloway researcher

Kings College London Current

Software development, security research, reverse engineering, bug bounty things.

Objective

How can you get a zero click on a iPhone?

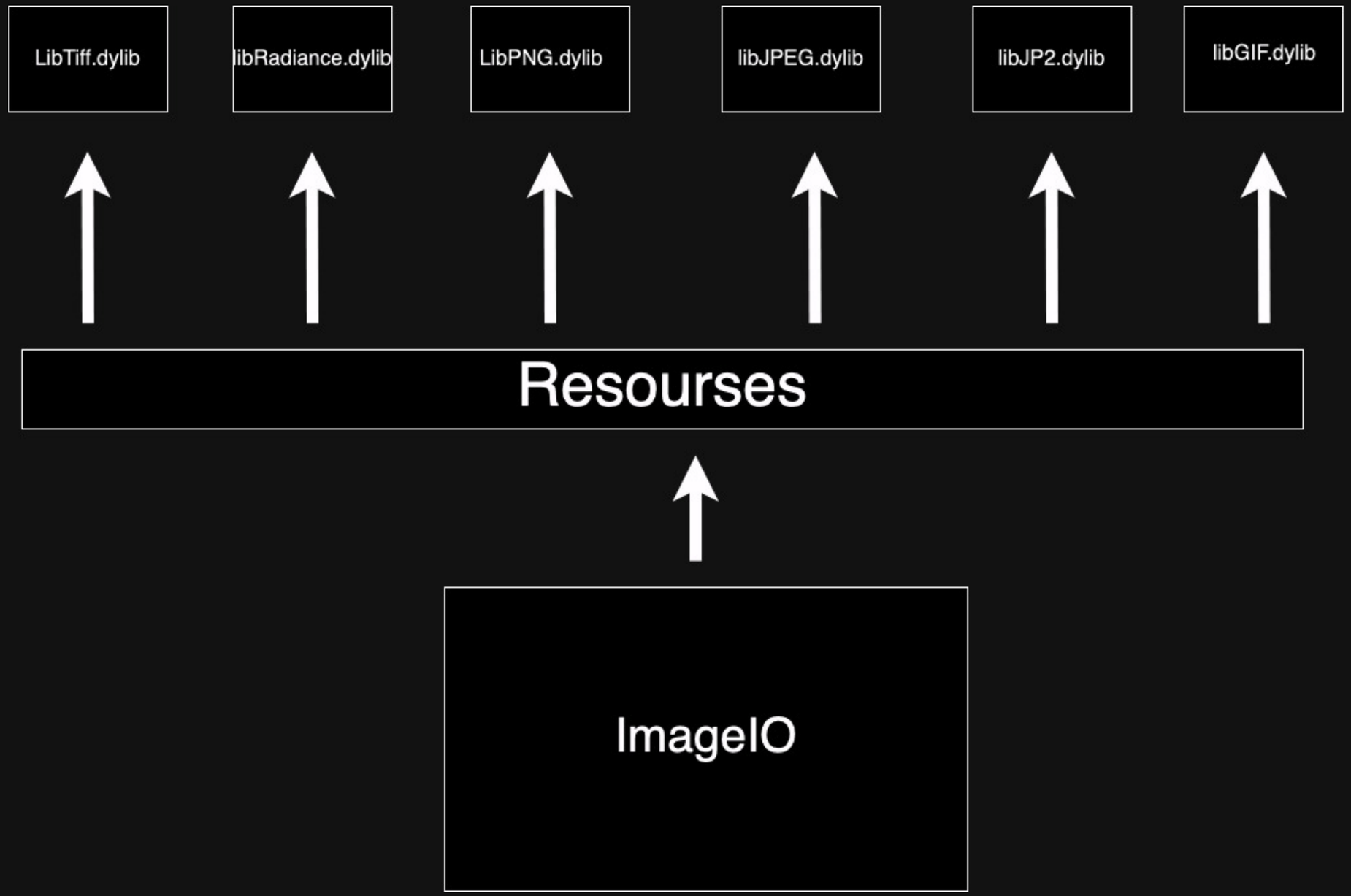
Send them a meme.jpg

Basic stack overview

Format Name	Is Supported by iOS	Is Supported by macOS
AI	✅	✅
ASTC	✅	✅
ATX	✅	✅
AppleJPEG	✅	✅
BC	✅	✅
BMP	✅	✅
CUR	✅	✅
ETC	✅	✅
GIF	✅	✅
HEIF	✅	✅
ICNS	✅	✅
ICO	✅	✅
JP2	✅	✅
KTX	✅	✅
KTX2	✅	✅
LibJPEG	✅	✅
MPO	✅	✅
OpenEXR	✅	✅
PBM	✅	✅
PDF	✅	✅
PNG	✅	✅
PICT	❌	✅
PSD	✅	✅
PVR	✅	✅
RAD	✅	✅
SGI	❌	✅
TGA	✅	✅
TIFF	✅	✅
WebP	✅	✅
Total Supported Formats:	27	29

Iphone Wiki Ioimage

ImageIO


LibTiff.dylib
libRadiance.dylib
LibPNG.dylib
libJPEG.dylib
libJP2.dylib
libGIF.dylib
Resourses
ImageIO

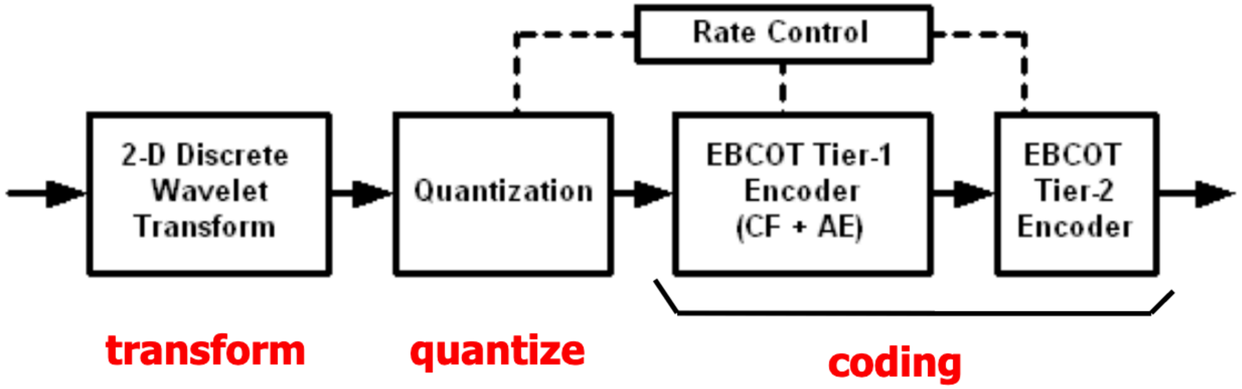
Jpeg2000

- **Improved Image Quality:** Provides superior image quality, preserving fine details.
- **Efficient Compression:** Achieves better compression, especially for high and low-frequency information.
- **Advanced Features:** Supports progressive coding and region of interest, catering to diverse applications.
- **Applications:** Developed to meet the evolving needs of industries like medical imaging, satellite imagery, and digital cinema.

Jpeg 2000

JPEG2000 Encoder Block Diagram

- Key Technologies:
  - Discrete Wavelet Transform (DWT)
  - Embedded Block Coding with Optimized Truncation (EBCOT)



Yu-Wei 2003,Still Image Compression jpeg & jpeg2000

LibJP2.dylib

Disclosed to Apple 27/01/2024

Somona 14.3

3428 functions for Image handling

Pointer dereference in kd_packet_sequencer::next_in_sequence

Pointer dereference in kd_tile::set_elements_of_interest

Reachable Assertion in kdu_kernels::derive_taps_and_gains

kd_packet_sequencer

```
int kd_packet_sequencer::next_in_sequence(kd_resolution*& resolution, kdu_coords& coords) {
    // Function signature: int kd_packet_sequencer::next_in_sequence(kd_resolution*&, kdu_coords&)
    // Save registers and allocate stack space
    stack[16] = r20;
    stack[24] = r19;
    stack[32] = r29;
    stack[40] = r30;
    // Get arguments
    void* kd_resolution = resolution; //arg0
    void* kdu_coords = coords; //arg1
    // Adjust stack pointer
    r31 = r31 + 0xfffffffffffffd0;
    // Check if the current packet is not the last one
    kd_resolution* r8 = *kd_resolution; //
    if (*(int32_t *)(r8 + 0x134) != *(int32_t *)(r8 + 0xd0)) {
        int32_t r16 = *(int32_t *)(kd_resolution + 0x20);

        // Check the value of r16
        if (r16 <= 0x4) {
            if (r16 <= 0x4) {
                // Check CPU_FLAGS & BE
                if (!(CPU_FLAGS & BE)) {
                    r16 = 0x0;
                } else {
                    r16 = r16;
                }
            }
            // Call a function based on the value of r16
            kd_resolution = (0x18eacd9c0 + sign_extend_64(*(int32_t *)(0x18eacdac0 + r16 * 0x4)))();
        } else {
            // Call kd_packet_sequencer::next_in_sequence()
            kd_packet_sequencer::next_in_sequence();
        }
    } else {
        // Set return value to 0 if the current packet is the last one
        kd_resolution = 0x0;
    }
    // Return the result
    return kd_resolution;
```

Possible Exploitation

```
// Assume an attacker has control over the kd_resolution object
kd_resolution* attacker_controlled_resolution = // Image Object our encoded data
// Modify the necessary fields within the attacker-controlled kd_resolution object
attacker_controlled_resolution->attacker_controlled_field = 0xfffffff;  // Modify fields based on the target system's memory layout
kd_packet_sequencer::next_in_sequence(attacker_controlled_resolution, ...);
kd_resolution* r8 = *kd_resolution;

// Check if the current packet is not the last one
if (*(int32_t *)(r8 + 0x134) != *(int32_t *)(r8 + 0xd0)) {
    int32_t r16 = *(int32_t *)(kd_resolution + 0x20);

    // Check the value of r16
    if (r16 <= 0x4) {
        if (r16 <= 0x4) {
            // Check CPU_FLAGS & BE
            if (!(CPU_FLAGS & BE)) {
                r16 = 0x0;
            } else {
                r16 = r16;
            }
        }
        // Call a function based on the value of r16
        kd_resolution = (0x18eacd9c0 + sign_extend_64(*(int32_t *)(0x18eacdac0 + r16 * 0x4)))();
    } else {
        // Call kd_packet_sequencer::next_in_sequence()
        kd_packet_sequencer::next_in_sequence();
    }
} else {
    // Set return value to 0 if the current packet is the last one
    kd_resolution = 0x0;
}

// Assuming an attacker-controlled value in r16, the attacker could manipulate the code flow.
// Example: Set r16 to a value corresponding to an attacker-controlled function pointer
r16 = attacker_controlled_resolution->attacker_controlled_field;

// Call a function based on the manipulated value of r16
kd_resolution = (0x18eacd9c0 + sign_extend_64(*(int32_t *)(0x18eacdac0 + r16 * 0x4)))();
// The attacker-controlled function pointer may point to shellcode or any arbitrary code.
```

CVE-2023-???????? Heap overflow in *ktxTexture2_constructFromStreamAndHeader*

2023-27939

Allocate memory on heap for ***r1*** **is memory of** ***r0***

Allocate memory on heap for ***r1***

Allocate memory on heap for ***r0***

Sst stream value

Funtion

ktxTexture2_constructFromStreamAndHeader

```
r1 = malloc(r24);
```

```
r1 = malloc(r23);
```

```
r0 = malloc(r24);
```

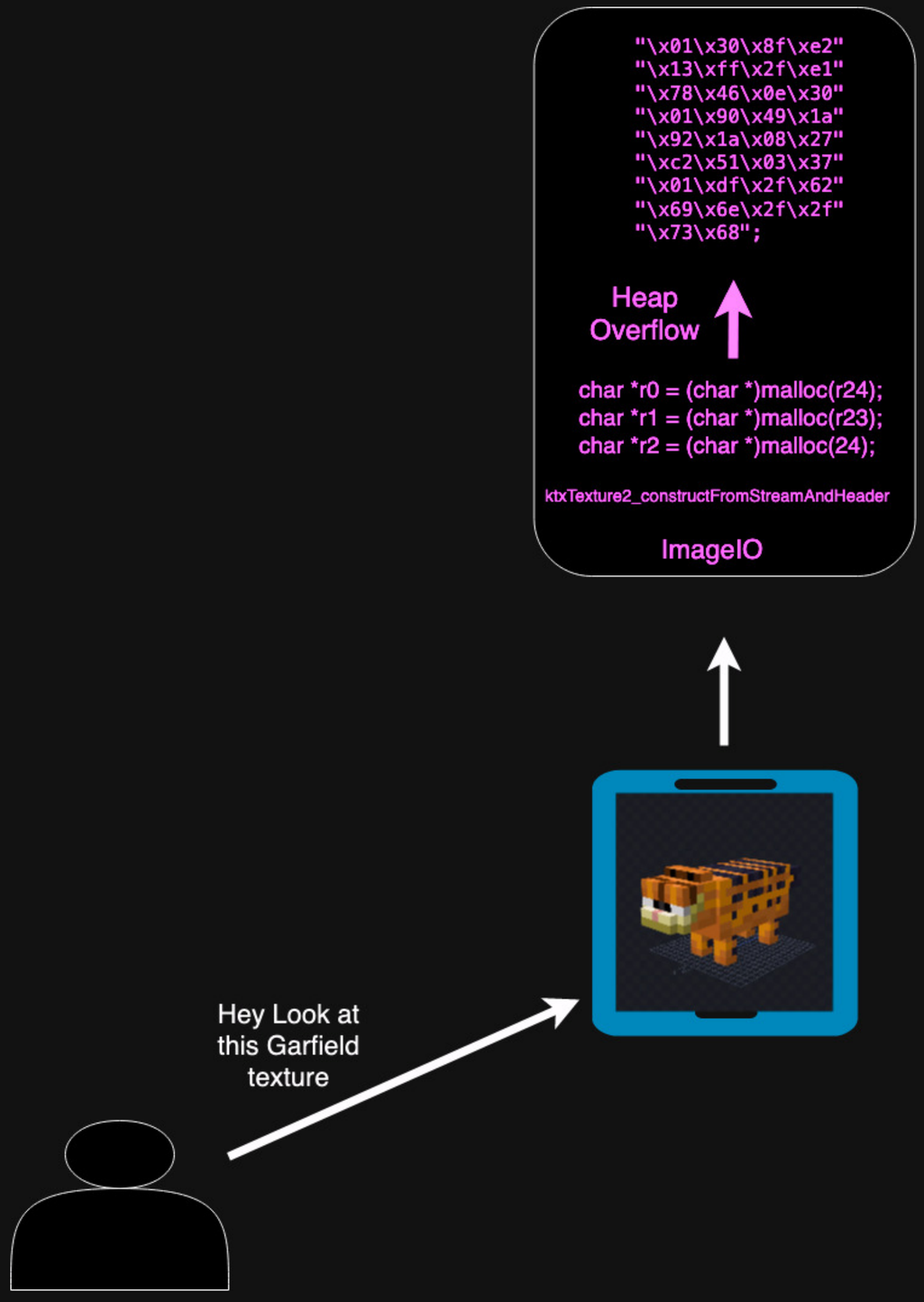
```
r24 = *(int32_t *)(r21 + 0x3c);
```

```
_ktxCheckHeader2_
```

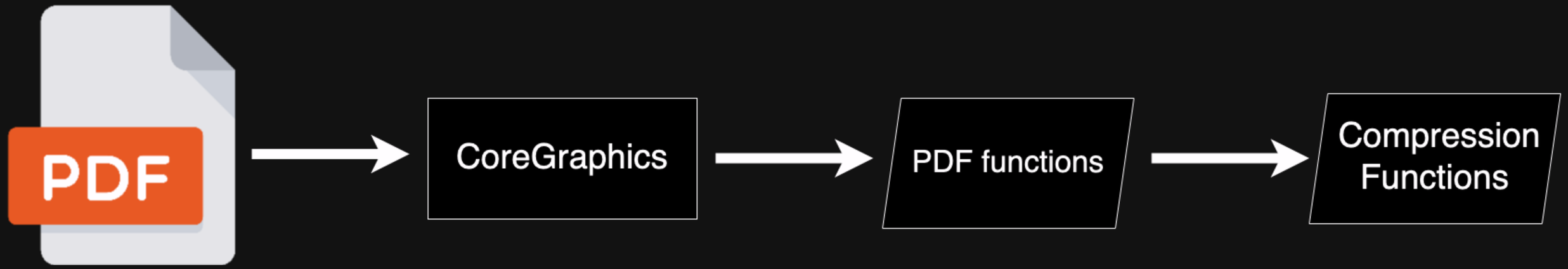
A Little deeper

```c
int main() {
    size_t r24 = 24;
    size_t r23 = 23;

    // Allocate memory for r0, r1, and r2
    char *r0 = (char *)malloc(r24);
    char *r1 = (char *)malloc(r23);
    char *r2 = (char *)malloc(24);

    // Assume that we copy some data into r1, and a vulnerability exists here
    // due to insufficient bounds checking.
    strcpy(r1, "This is a buffer overflow example");

    // Now, an attacker might overflow r1 to overwrite control data in r2
    // Let's assume r2 has some critical control data like function pointers or return addresses.

    // Example shellcode (x86 assembly) that an attacker might inject:
    char shellcode[] =
        "\x31\xc0"                  // xor eax, eax (clear eax register)
        "\x50"                      // push eax (push null-terminated string terminator)
        "\x68\x2f\x2f\x73\x68"      // push 0x68732f2f (push "//sh" onto the stack)
        "\x68\x2f\x62\x69\x6e"      // push 0x6e69622f (push "/bin" onto the stack)
        "\x89\xe3"                  // mov ebx, esp (set ebx to the address of the stack)
        "\x50"                      // push eax (push null for termination)
        "\x53"                      // push ebx (push the address of "/bin//sh" onto the stack)
        "\x89\xe1"                  // mov ecx, esp (set ecx to the address of the stack)
        "\x31\xd2"                  // xor edx, edx (clear edx register)
        "\xb0\x0b"                  // mov al, 0x0b (set syscall number for execve)
        "\xcd\x80";                 // int 0x80 (make the system call)

    // Overflow r1 to overwrite data in r2 with the shellcode
    memcpy(r1, shellcode, sizeof(shellcode));

    // The attacker's goal is to manipulate control data in r2 so that the program
    // will execute the injected shellcode when r2 is used.

    // Free allocated memory
    free(r0);
    free(r1);
    free(r2);

    return 0;
}
```

Possible Example



CoreGraphics



J2BIG

J2BIG:

- J2BIG is associated with JPEG 2000 and specifically handles bi-level images (binary images with two colors, usually black and white).

Compression Process:

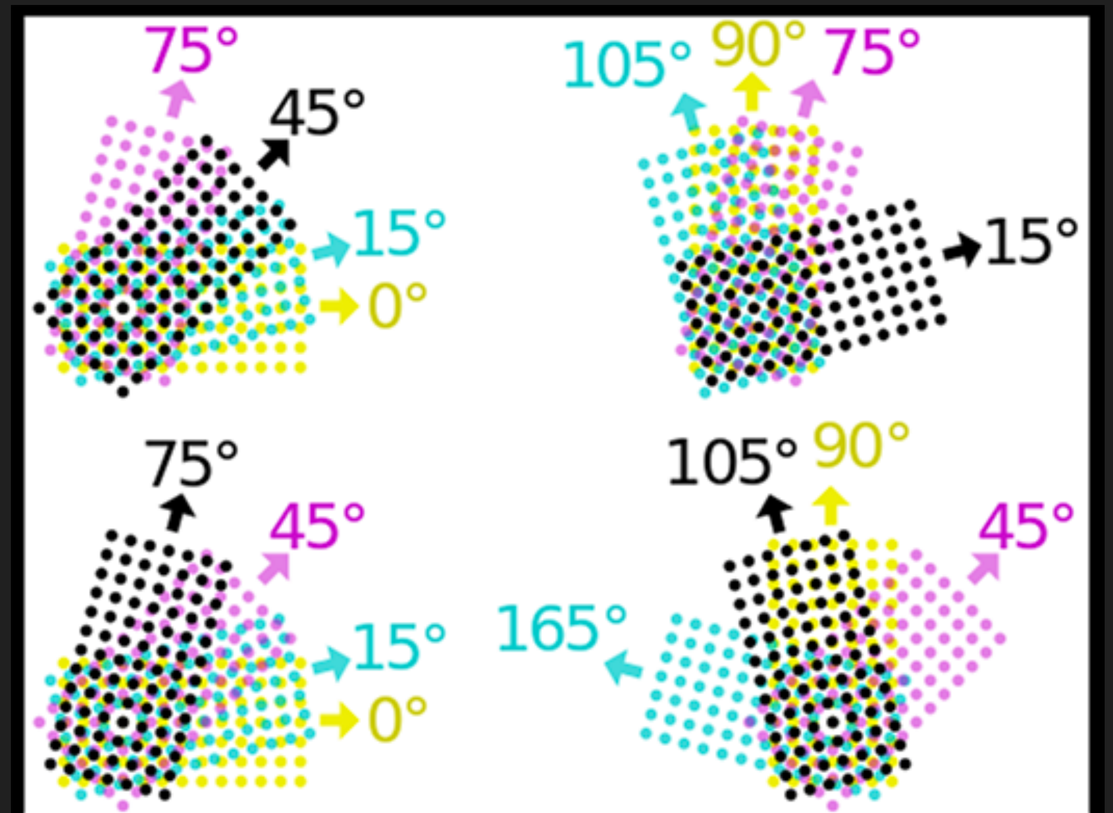
- JPEG 2000 compression involves multiple steps, including wavelet transformation, quantization, and entropy coding.

LSB and Transformations:

- The concept of taking the Least Significant Bit (LSB) for transformations is not directly associated with JPEG 2000.

Efficient Binary Image Compression:

- J2BIG within JPEG 2000 is designed to efficiently compress binary images, such as those with only two colors (black and white).

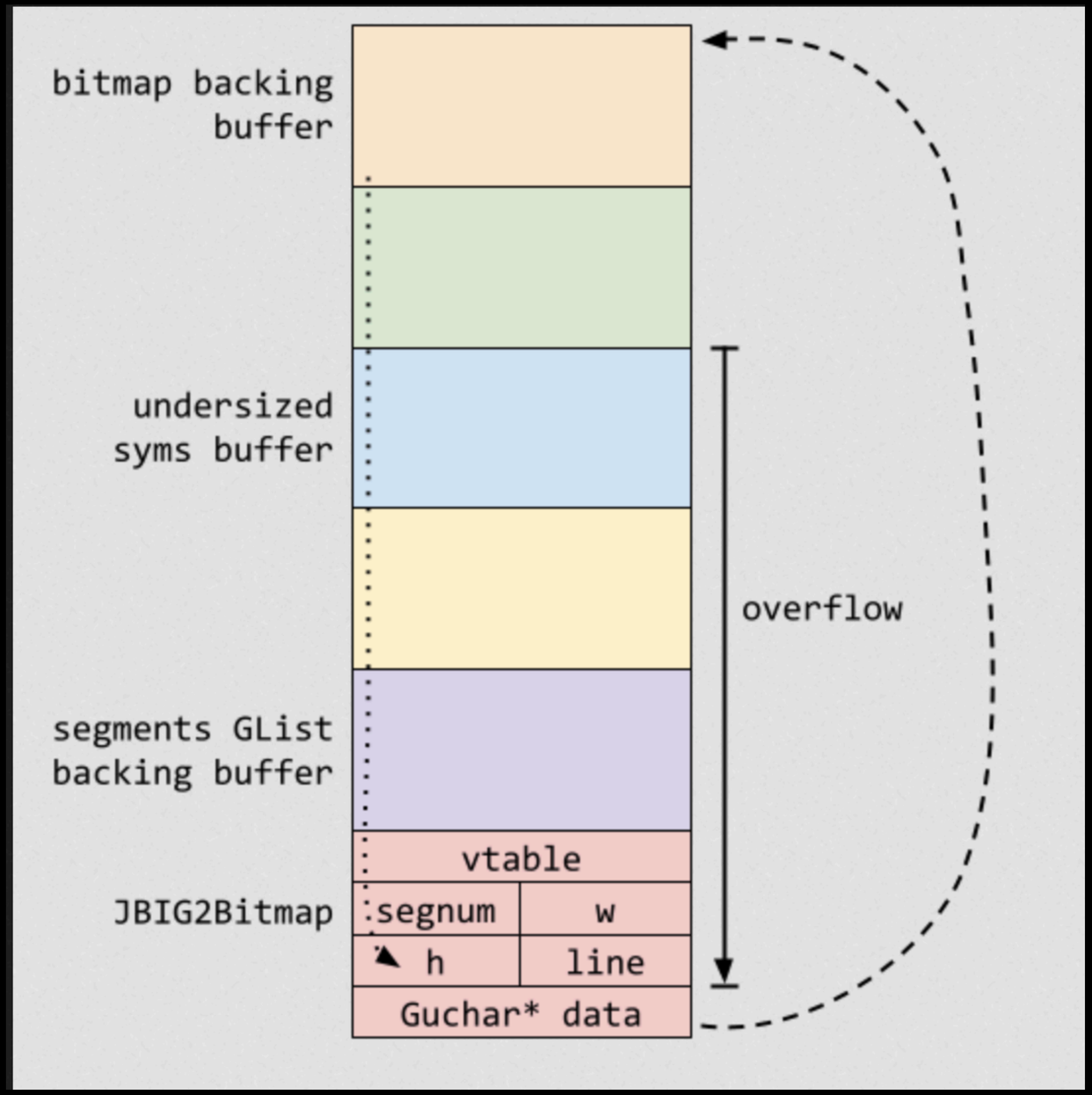


Halftone encoding

NSO Group

Overflow and exploitation

```
  Guint numSyms; // (1)

  numSyms = 0;
  for (i = 0; i < nRefSegs; ++i) {
    if ((seg = findSegment(refSegs[i]))) {
      if (seg->getType() == jbig2SegSymbolDict) {
        numSyms += ((JBIG2SymbolDict *)seg)->getSize();  // (2)
      } else if (seg->getType() == jbig2SegCodeTable) {
        codeTables->append(seg);
      }
    } else {
      error(errSyntaxError, getPos(),
            "Invalid segment reference in JBIG2 text region");
      delete codeTables;
      return;
    }
  }
...
  // get the symbol bitmaps
  syms = (JBIG2Bitmap **)gmallocn(numSyms, sizeof(JBIG2Bitmap *)); //
(3)

  kk = 0;
  for (i = 0; i < nRefSegs; ++i) {
    if ((seg = findSegment(refSegs[i]))) {
      if (seg->getType() == jbig2SegSymbolDict) {
        symbolDict = (JBIG2SymbolDict *)seg;
        for (k = 0; k < symbolDict->getSize(); ++k) {
          syms[kk++] = symbolDict->getBitmap(k); // (4)
        }
      }
    }
  }
```



Pegasus Spyware

Image based chained exploits

Turing complete machine

At least 30 journalists, lawyers and activists hacked with Pegasus in Jorda forensic probe finds

A new report says Israeli-made Pegasus spyware was used in Jordan to hack the cellphones of nearly thre dozen people including journalists, lawyers and human rights activists

Frank Bajak • 3 hours ago

Jamal Khashoggi's widow files lawsuit against NSO Group

Hanan Elatr Khashoggi says Israeli tech company used its Pegasus

Investigation reveals Pegasus spyware used to track over 50,000 people

July 19, 2021

Thank you, and any Questions?