Weekly Notes for CTF 🚩

Week 4

Astrageldon

2023-12-15

~~*[REDACTED]*~~ 竟然说我的笔记很干 🤔 那怎么办好呢

对了，那就叫几辆消防车来洒点水吧 🥳

哔嘟哔嘟 🚒 🚒 🚒 🚒 🚒 🚒 🚒 哔嘟哔嘟

💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦

💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦

💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦

💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦 💧 💦 💦

1 Misc

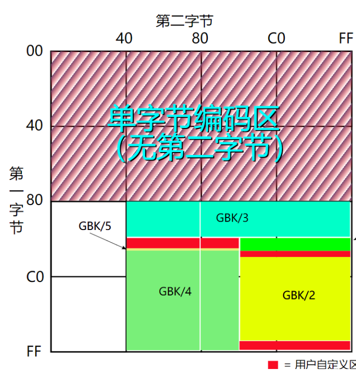
1.1 GBK Mojibake

有时，我们会神奇地发现，一些本应显示中文的地方出现了一堆奇奇怪怪的字符。

这些字符时常是由拉丁字母以及一些数字符号构成的~~（看上去很神秘）~~。我们注意到 GBK 的编码方式：

GBK的編碼範圍

範圍	第1位元組	第2位元組	編碼數	字數
水準GBK/1	A1 – A9	A1 – FE	846	717
水準GBK/2	B0 – F7	A1 – FE	6,768	6,763
水準GBK/3	81 – A0	40 – FE（7F 除外）	6,080	6,080
水準GBK/4	AA – FE	40 – A0（7F 除外）	8,160	8,160
水準GBK/5	A8 – A9	40 – A0（7F 除外）	192	166
用戶定義	AA – AF	A1 – FE	564	
用戶定義	F8 – FE	A1 – FE	658	
用戶定義	A1 – A7	40 – A0（7F 除外）	672	
合計：			23,940	21,886

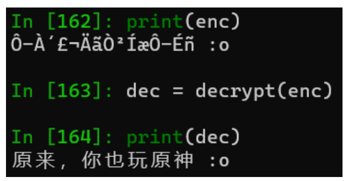


对于常用字而言，一般分布于 GBK/2 的范围内，而对于 Unicode 码而言 0xA1~0xFE 范围内的字符全部都是可打印字符。于是我们将一个汉字的 GBK 编码拆成两个分开的字节，使用 UTF-8 对这两个字节分别解码，就会产生如下的乱码（文字化け）效果：

而对这样的乱码进行解码也十分容易，将两个一字节的 Unicode 码拼起来，再使用 GBK 编码解码即可：

```python
def decrypt(c):
    return bytes.fromhex(''.join([hex(ord(x))[2:].zfill(2) for
        x in c])).decode('gbk')
```

效果如下：



💧💦💦💧💦💦💧💦💦💧💦💦💧💦💦💧💦💦

1.2 Discrete Cosine Transform

众所周知，

$$\boldsymbol{X} = \mathcal{F}\{\boldsymbol{x}\}, \qquad X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N}kn}$$

$$\boldsymbol{x} = \mathcal{F}^{-1}\{\boldsymbol{X}\}, \qquad x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k \cdot e^{\frac{2\pi i}{N}kn}$$

对序列 $\{x_n\}$ 进行偶延拓，就可以去掉虚部：

$$\boldsymbol{X} = \mathcal{F}\{\boldsymbol{x}\}, \qquad X_k = 2\sum_{n=0}^{N-1} x_n \cdot \cos\frac{(n+1/2)\pi k}{N}$$

这样就可以避免对复数的讨论。将基底归一化：

$$X_k = c(k)\sum_{n=0}^{N-1} x_n \cdot \cos\frac{(n+1/2)\pi k}{N}, \qquad c(k) = \begin{cases} \sqrt{1/N}, & k=0 \\ \sqrt{2/N}, & \text{Otherwise} \end{cases}$$

在图像处理的过程中，使用的是二维的离散余弦变换，它是二维离散傅里叶变换的变体：

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x_{n_1,n_2} \cdot \cos\frac{(n_1+1/2)\pi k}{N_1}\cos\frac{(n_2+1/2)\pi k}{N_2}$$

归一化后：

$$X_{k_1,k_2} = c_1(k_1)c_2(k_2)\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x_{n_1,n_2} \cdot \cos\frac{(n_1+1/2)\pi k_1}{N_1}\cos\frac{(n_2+1/2)\pi k_2}{N_2}$$

$$x_{n_1,n_2} = \sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1} c_1(k_1)c_2(k_2)X_{k_1,k_2} \cdot \cos\frac{(n_1+1/2)\pi k_1}{N_1}\cos\frac{(n_2+1/2)\pi k_2}{N_2}$$

一般 $N_1 = N_2$，于是可以将其表示为正交矩阵诱导的合同变换：

$$\boldsymbol{X} = \boldsymbol{C}^T\boldsymbol{x}\boldsymbol{C}, \qquad \boldsymbol{x} = \boldsymbol{C}\boldsymbol{X}\boldsymbol{C}^T$$

二维离散余弦变换的意义在于，将一个块中的的图像从空间域转换为频域，也即，将其分解为若干个频率不同的横向与纵向的波。于是，我们可以对图像中不同空间频率的部分进行调整，弱化人眼不容易识别的高频部分，从而达到图片压缩的效果。

1.3 B&W JPEG Compression

JPEG 压缩算法的核心在于，剪去图像中人眼难以识别的高频部分，再将剩下的非零空间频率系数保存下来。

1.3.1 Blockwise DCT

将一幅黑白图像裁剪为若干个 8×8 大小的子块，对于每一个子块，每个数字减去 128 以后进行离散余弦变换操作。此时，左上角的 DCT 系数 $X(0,0)$ 被称作直流（Direct Current, DC）系数，其余的 63 个系数被称作交流（Alternating Current, AC）系数。该操作在浮点数的精度足够高时是无损的。

1.3.2 Quantization

量化是一个有损的过程，首先我们计算如下矩阵

$$Q(u,v)=\begin{cases}\max\{\left\lfloor\left(2-\frac{\mathrm{QF}}{50}\right)Q_0(u,v)+\frac{1}{2}\right\rfloor,1\}, & 50\le \mathrm{QF}\le 100\\ \left\lfloor\frac{50}{\mathrm{QF}}Q_0(u,v)+\frac{1}{2}\right\rfloor, & 0<\mathrm{QF}<50\end{cases}$$

其中 QF 是质量因子（Quality Factor），Q_0 是亮度的标准量化矩阵

$$Q_0=\begin{pmatrix}16&11&10&16&24&40&51&61\\12&12&14&19&26&58&60&55\\14&13&16&24&40&57&69&56\\14&17&22&29&51&87&80&62\\18&22&37&56&68&109&103&77\\24&35&55&64&81&104&113&92\\49&64&78&87&103&121&120&101\\72&92&95&98&112&100&103&99\end{pmatrix}$$

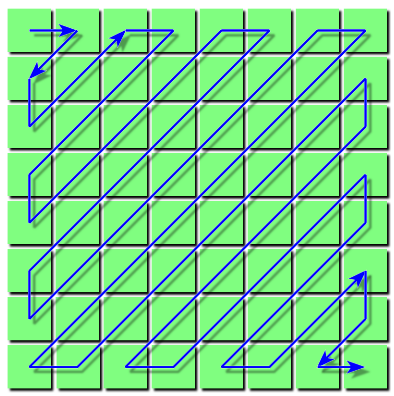
对 DCT 后得到的矩阵逐项除以 Q 中的对应项，四舍五入，便可以得到

形状如下的量化后的矩阵

$$\begin{pmatrix} * & \cdots & * & 0 & \cdots & 0 \\ \vdots & \iddots & 0 & 0 & \cdots & 0 \\ * & 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 0 \end{pmatrix}$$

1.3.3 Entropy Coding

现在，按照下图中的 Z 字形顺序（zig-zag）将上一节中量化后的矩阵排列成一个序列。



使用哈夫曼编码（Huffman coding）将该序列存储下来即可，这样的存储方式可以节省不少的空间。和文本压缩存储类似的是，标志着字符串结尾的 EOF 在这里是用来标记后续序列中数字全为零的 EOB（End of Block）。

1.3.4 Decompression

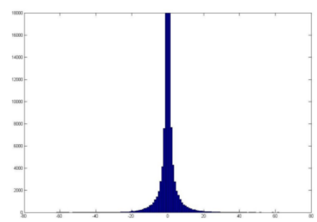
将上述过程反过来即可，需要注意的是最后需要对每个数字进行四舍五入。

1.4 DCT Steganography

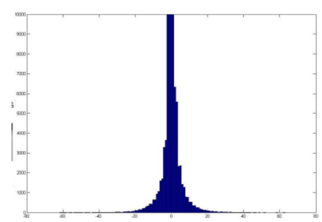
众所周知，著名的 LSB（Least Significant Bit）隐写是在图像的空间域上隐藏数据，然而，对于一般的 LSB 隐写而言，通过将一幅图片各个像素的 LSB 位描绘出来便可以发现是否存在这种隐写方式，并且图片压缩后密文数据便会损坏，而将密文隐藏在频域则可以尽量避开这些缺点。

我们可以将密文嵌入量化后的矩阵中除了 $-1, 0, 1$ 以外的 AC 系数的最低位（频域上的 LSB 隐写）。也可以控制某两个 AC 系数的大小关系，为了使这种大小关系在量化前后尽可能地一致，我们需要寻找量化表中的两个值接近并且较小的项，就比如 $Q(5,2)$ 和 $Q(4,3)$，$Q(4,1)$ 和 $Q(3,2)$，$Q(1,2)$ 和 $Q(3,0)$。

频域上 LSB 隐写的检测方法：对于原始图像而言，DCT 系数分布的直方图比较对称，而对于一般的 DCT 隐写而言，图像会稍有偏移。更精确的判据可以用 χ^2 测试来刻画~~（挖坑）~~。



DCT Coefficients of unaltered JPEG



DCT Coefficients of steganographed JPEG

2 Crypto

2.1 Hensel's Lifting Lemma

💧💦💦💧💦💦💧💦💦💧💦💦💧💦💦💧💦💦💧💦💦

Theorem. 若 $f(x) \in \mathbb{Z}[x]$，$p \in \mathbb{P}, x, s \in \mathbb{Z}$，$p^s \mid f(x), \gcd(f'(x), p) = 1$，则 $\exists x' \in \mathbb{Z}, x' \equiv x \pmod{p^s}$，且

$$x' \equiv x - uf(x) \pmod{p^{s+1}}, \quad u \in \mathbb{Z}, \quad uf'(x) \equiv 1 \pmod{p}$$

该定理可以在已知 $f(x) \equiv 0 \pmod{p^k}$ 的解的情况下获得 $f(x) \equiv 0 \pmod{p^{k+1}}$ 的解。证明的思路是对 f 使用泰勒公式：

$$x' = x + tp^s$$

$$\begin{aligned} f(x') = f(x) + f'(x)tp^s + K_1 \cdot p^{s+1}, &\qquad K_1 \in \mathbb{Z} \\ \Rightarrow f(x') \equiv 0 \pmod{p^{s+1}} & \\ \Leftrightarrow tf'(x)p^s = -f(x) + K_2 \cdot p^{s+1}, &\qquad K_2 \in \mathbb{Z} \\ \Leftrightarrow t \equiv -\frac{uf(x)}{p^s} \pmod{p} & \end{aligned}$$

□

2.2 Smart's Attack SPEEDRUN!! Really Speeeedy :p

设 $\#E(\mathbb{F}_p) = p$，而 $\psi_p(S) = -\frac{x(S)}{y(S)}$ 是由 $E_1(\mathbb{Q}_p)$ 到 $p\mathbb{Z}_p$ 的同构，其中 $E_1(\mathbb{Q}_p) = \mathrm{Ker}\varphi$，而同态 $\varphi : E(\mathbb{Q}_p) \to E(\mathbb{F}_p)$ 被定义为将点的每个分量对 p 取模。

于是由 $Q = kP,\ Q, P \in E(\mathbb{F}_p)$ 立即得到

$$k = \frac{\psi_p(pQ')}{\psi_p(pP')}$$

其中 Q', P' 为 Q, P“lift”至 $E(\mathbb{Q}_p)$ 后得到的点。