ippatsu-nyuukon

发表于 2021-01-21 分类于 Challenge， 2017， HCTF， Bin
Challenge | 2017 | HCTF | Bin | ippatsu-nyuukon

点击此处获得更好的阅读体验

WriteUp 来源

https://xz.aliyun.com/t/1589

题目考点

- *驱动逆向*

解题思路

0x00 写在前面

设计思路：应用层与驱动层通信，在驱动层加密由应用层发送过来的明文后比较flag，并输出结果

部分细节：

- *驱动层的分发函数分为2部分，SEND和RECV；*
- *SEND：接受从应用层发来的明文并加密，其本体是DES*
- *RECV：比较加密后的明文和加密flag*
- *驱动层接受的明文，实际上只有第一次加密结果是正确的*
- *DES后将不可视数据转为hex*
- *加密数据与加密flag比较前先异或同一个随机字节*

0x01 wp

跟到分发函数的SEND, 定位加密算法
因为DES对称加密算法，所以从ida中抠出来，修改小部分并添加密文+key就可以跑解密脚本了

```
// desrypt_des.cpp
#include <stdio.h>
#include <string.h>

#define maxn 0x8000     // 理论支持明文长度
//#define ENCODE 0,16,1       // 加密用的宏
#define DECODE 15,-1,-1     // 解密用的宏

// 明文初始置换
char msg_ch[64] = {
    58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,
    57, 49, 41, 33, 25, 17,  9, 1, 59, 51, 43, 35, 27, 19, 11, 3,
    61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7
};

// 密钥初始置换
char key_ch[56] = {
    57, 49, 41, 33, 25, 17,  9,  1, 58, 50, 42, 34, 26, 18,
    10,  2, 59, 51, 43, 35, 27, 19, 11,  3, 60, 52, 44, 36,
    63, 55, 47, 39, 31, 23, 15,  7, 62, 54, 46, 38, 30, 22,
    14,  6, 61, 53, 45, 37, 29, 21, 13,  5, 28, 20, 12,  4
};

// 扩展置换
char msg_ex[48] = {
```

```
    32,  1,  2,  3,  4,  5,  4,  5,  6,  7,  8,  9,
    8,  9, 10, 11, 12, 13, 12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21, 20, 21, 22, 23, 24, 25,
    24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32,  1
};

// 每轮密钥的位移
char key_mov[16] = {
    1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
};

// 压缩置换
char key_cmprs[48] = {
    14, 17, 11, 24,  1,  5,  3, 28, 15,  6, 21, 10,
    23, 19, 12,  4, 26,  8, 16,  7, 27, 20, 13,  2,
    41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48,
    44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32
};

// S 盒置换
char s_box[8][6][16] = {
    // S1
    14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9,  0,  7,
    0, 15,  7,  4, 14,  2, 13,  1, 10,  6, 12, 11,  9,  5,  3,  8,
    4,  1, 14,  8, 13,  6,  2, 11, 15, 12,  9,  7,  3, 10,  5,  0,
    15, 12,  8,  2,  4,  9,  1,  7,  5, 11,  3, 14, 10,  0,  6, 13,
    // S2
    15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2, 13, 12,  0,  5, 10,
    3, 13,  4,  7, 15,  2,  8, 14, 12,  0,  1, 10,  6,  9, 11,  5,
    0, 14,  7, 11, 10,  4, 13,  1,  5,  8, 12,  6,  9,  3,  2, 15,
    13,  8, 10,  1,  3, 15,  4,  2, 11,  6,  7, 12,  0,  5, 14,  9,
    // S3
    10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,  7, 11,  4,  2,  8,
    13,  7,  0,  9,  3,  4,  6, 10,  2,  8,  5, 14, 12, 11, 15,  1,
    13,  6,  4,  9,  8, 15,  3,  0, 11,  1,  2, 12,  5, 10, 14,  7,
    1, 10, 13,  0,  6,  9,  8,  7,  4, 15, 14,  3, 11,  5,  2, 12,
    // S4
    7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,  5, 11, 12,  4, 15,
    13,  8, 11,  5,  6, 15,  0,  3,  4,  7,  2, 12,  1, 10, 14,  9,
    10,  6,  9,  0, 12, 11,  7, 13, 15,  1,  3, 14,  5,  2,  8,  4,
    3, 15,  0,  6, 10,  1, 13,  8,  9,  4,  5, 11, 12,  7,  2, 14,
    // S5
    2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3, 15, 13,  0, 14,  9,
    14, 11,  2, 12,  4,  7, 13,  1,  5,  0, 15, 10,  3,  9,  8,  6,
    4,  2,  1, 11, 10, 13,  7,  8, 15,  9, 12,  5,  6,  3,  0, 14,
    11,  8, 12,  7,  1, 14,  2, 13,  6, 15,  0,  9, 10,  4,  5,  3,
    // S6
    12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11,
    10, 15,  4,  2,  7, 12,  9,  5,  6,  1, 13, 14,  0, 11,  3,  8,
    9, 14, 15,  5,  2,  8, 12,  3,  7,  0,  4, 10,  1, 13, 11,  6,
    4,  3,  2, 12,  9,  5, 15, 10, 11, 14,  1,  7,  6,  0,  8, 13,
    // S7
    4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,  7,  5, 10,  6,  1,
    13,  0, 11,  7,  4,  9,  1, 10, 14,  3,  5, 12,  2, 15,  8,  6,
    1,  4, 11, 13, 12,  3,  7, 14, 10, 15,  6,  8,  0,  5,  9,  2,
    6, 11, 13,  8,  1,  4, 10,  7,  9,  5,  0, 15, 14,  2,  3, 12,
    // S8
    13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3, 14,  5,  0, 12,  7,
    1, 15, 13,  8, 10,  3,  7,  4, 12,  5,  6, 11,  0, 14,  9,  2,
    7, 11,  4,  1,  9, 12, 14,  2,  0,  6, 10, 13, 15,  3,  5,  8,
    2,  1, 14,  7,  4, 10,  8, 13, 15, 12,  9,  0,  3,  5,  6, 11
};

// P 盒置换
char p_box[32] = {
    16, 7, 20, 21, 29, 12, 28, 17, 1,  15, 23, 26, 5,  18, 31, 10,
    2,  8, 24, 14, 32, 27, 3,  9,  19, 13, 30, 6,  22, 11, 4,  25
};

// 末置换
char last_ch[64] = {
    40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31,
    38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29,
    36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27,
    34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41,  9, 49, 17, 57, 25
};
```

```
103
104 // hash 置换，将加密后的密文置换为可读明文
105 char hs_ch[20] = "0123456789abcdef";
106 char sh_ch[128];
107
108 void init_trans() {
109     char i;
110     for (i = 0; i < 16; i++)
111         sh_ch[hs_ch[i]] = i;    // 完成hash转换的对应
112 }
113
114 char msg[maxn] = "aed3899df15bd7babb99acf5ebb9f5cd8cd44a77c53263de46ef9f3d773fe908";
115 char res[32];
116 char msgb[72], msgbt[72], keyb[18][72];
117 char key[16] = "deadbeef";
118
119 // 字符转成二进制
120 void ChToBit(char* dest, char* src, int length) {
121     int i, j;
122     char t;
123     for (i = 0; i < length; i++) {
124         for (j = 8, t = src[i]; j > 0; j--) {
125             dest[(i << 3) + j] = t & 1;   // 取字符末位
126             t >>= 1;
127         }
128     }
129 }
130
131 // 二进制转成字符
132 void BitToCh(char* dest, char* src, int length) {
133     int i;
134     for (i = 0; i < length << 3; i++) {
135         dest[i >> 3] <<= 1;
136         dest[i >> 3] |= src[i + 1];   // 添加到末位
137     }
138     dest[length] = 0;
139 }
140
141 // 批置换，以offset为偏移，以count为长度
142 void BatchSet(char* dest, char* src, char* offset, int count) {
143     int i;
144     for (i = 0; i < count; i++)
145         dest[i + 1] = src[offset[i]];
146 }
147
148 // 得到16轮所需的密钥
149 void getKeys() {
150     char tk[128], bk[72];
151     char* ptk = tk;
152     int i, j;
153     for (i = 0; i < 8; i++)
154         key[i] <<= 1; // 跳过奇偶校验位
155     ChToBit(bk, key, 8);
156     BatchSet(tk, bk, key_ch, 56);
157     for (i = 0; i < 16; i++) {
158         for (j = 0; j < key_mov[i]; j++, ptk++) {
159             ptk[57] = ptk[28];
160             ptk[28] = ptk[1];
161         }
162         BatchSet(keyb[i], ptk, key_cmprs, 48);
163     }
164 }
165
166 // 将密文转换为真正的密文
167 void dropMsg(char* dest, char* src) {
168     int i;
169     for (i = 0; i < 16; i++) {
170         dest[i >> 1] = (dest[i >> 1] << 4) | sh_ch[src[i]];
171     }
172 }
173
174 void DES(char* pmsg, int st, int cl, int step) {
175     int i, row, col;
176     char r[64], rt[48], s[8];
177     ChToBit(msgbt, pmsg, 8);
178     BatchSet(msgb, msgbt, msg_ch, 64); // 初始置换
```

```
    for (; st != cl; st += step) {
        memcpy(rt, msgb + 33, 32);
        BatchSet(r, msgb + 32, msg_ex, 48); // 扩展置换
        for (i = 1; i <= 48; i++)
            r[i] ^= keyb[st][i]; // 异或操作
                                 // s_box 代替
        for (i = 0; i < 48; i += 6) {
            row = col = 0;
            row = r[i + 1] << 1 | r[i + 6];
            col = (r[i + 2] << 3) | (r[i + 3] << 2) | (r[i + 4] << 1) | r[i + 5];
            s[i / 12] = (s[i / 12] <<= 4) | s_box[i / 6][row][col];
        }
        ChToBit(r, s, 4);
        BatchSet(msgb + 32, r, p_box, 32); // p_box 置换
        for (i = 1; i <= 32; i++)
            msgb[i + 32] ^= msgb[i]; // 异或
        memcpy(msgb + 1, rt, 32);
    }
    memcpy(msgbt + 33, msgb + 1, 32);
    memcpy(msgbt + 1, msgb + 33, 32);
    BatchSet(msgb, msgbt, last_ch, 64); // 末置换
    BitToCh(res, msgb, 8); // 转为原明文
}

int main(int arg, char* arv[]) {
    init_trans();
    char mode = 'd';

    getKeys(); // 得到16轮要用到的密钥

    int i;

    printf("dec: ");
    for (i = 0; msg[i]; i += 16) {
        dropMsg(res, msg + i); // 将密文转换为真正的密文
        DES(res, DECODE); // 解密
        printf("%s", res);
    }
    printf("\n");
    return 0;
}
```

简单来说 只要把加密宏[0, 16 ,1] 替换为[15, -1, -1] 解密宏即可。

附一张成功cm的截图

Flag

```
hctf{Dr1v5r_M5ngM4n_2Oi7}
```

- ***本文作者：*** *CTFHub*
- ***本文链接：*** *https://writeup.ctfhub.com/Challenge/2017/HCTF/Bin/pbWqDHo9FzLyWJpqC6jFR9.html*
- ***版权声明：*** *本博客所有文章除特别声明外，均采用 BY-NC-SA 许可协议。转载请注明出处！*

 # Challenge # 2017 # HCTF # Bin
babystack
babyre