BabyRE:

Đây là file thực thi window 32 bit.



Load file vào ida32, sau đó F5 để gen ra pseudocode. Pseudocode sau khi đã thực hiện nắn chỉnh các hàm và kiểu dữ liệu:

Application, timeline

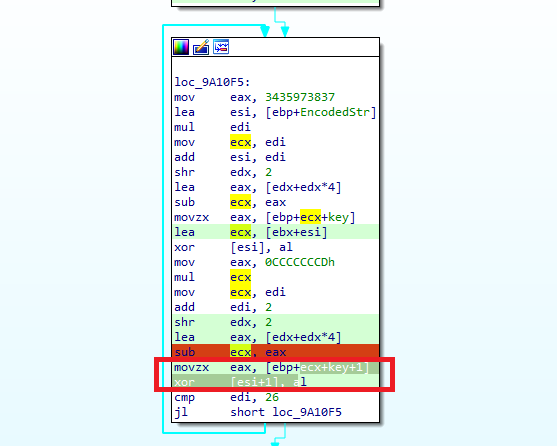
Description automatically generated

Ở trong đây có 2 biến quan trọng, đó là EncodedStr(payload): là 1 chuỗi byte 26 kí tự được chương trình khởi tạo sẵn và key là một chuỗi byte 5 kí tự, được người dùng nhập vào (dòng 14, 15 cho ta biết điều đó).

Từ 16-22 cho thấy chương trình thực hiện xor decode string đã khởi tạo ban đầu, sử dụng 5 byte đã nhập vào để làm key. Do ta biết trước 5 byte đầu của output là “flag{” nên có thể sử dụng dữ kiện này để mò lấy key. Nếu dữ kiện này là không đủ thì buộc phải brute-force.

Trong phần xor decode, 2 byte của payload được xử lý 1 lần (Do j+=2 và trong for thực hiện load EncodedStrp[j] và EncodedStrp[j+1]). Trong 2 byte đó, byte thứ nhất được xor với byte thứ Z của key, Z = j%5. Như vậy, với j = 0, 2, 4 thì byte thứ 0, 2, 4 của payload được xor với byte thứ 0, 2, 4 của key, kết quả lần lượt là ‘f’(0x66), ‘a’(0x61), ‘{‘(0x7B). Để tìm key ta chỉ cần xor ngược lại. Vậy đã tìm được byte 0, 2, 4 của key: 4, 15, 42

Còn byte thứ 1 và 3 của key. Dòng 21 thực hiện xor để decode 2 byte này. Biểu thức \_j\_ - 5 \* (&v5[1 - EncodedStr] / 5) + 1 khá khó tính nên chúng ta sẽ không tính mà thực hiện debug để xem giá trị của nó.



Ta có thể thấy 1 byte của payload được xor với 1 byte của key. Byte thứ ecx + 1 của key sẽ được load vào eax. Ta sẽ breakpoint tại đây để xem ecx có giá trị bao nhiêu.

Kết quả:

Lần hit đầu tiên: ecx = 0, như vậy byte 1 của key được load.

Graphical user interface, application, Word

Description automatically generated

Và lần hit thứ hai: ecx = 2, như vậy byte thứ 3 của key được load

Graphical user interface, application, Word

Description automatically generated

Tương tự như trên ta tính được byte 1 và 3 của key là 8, 23

Như vậy ta có key là 4, 8, 15, 23, 42. Nhập key vào và ta có flag của bài:

flag{easy\_baby\_challenge}

