

Forum:[Hack Forums](#)

Thread: [My file crypter has no trouble crypting text file - but fails at binaries?](#)

pebblak - September 4, 2019, 11:31 am

Hey all. I'm really new to C and crypting binaries for that matter so I'm probably missing something obvious.

The issue I'm having is, my file crypter can encrypt and decrypt sample text files with no problem (files containing a line of text), however when tried on a binary, the decrypted binary throws this:

Code ./unencrypted_test_program: line 1: \$'\177ELF\002\001\001': command not found
The plain binary is, in fact, 12 KB in size. The encrypted version is 7 bytes, and so is the decrypted version of the encrypted binary. Naturally this doesn't make sense, because I'm using a simple XOR.

Here is my code:

```
Code int xorFile(char *infile, char *outfile){
    FILE *fp;
    FILE *fp2;
    char buf[20000];

    fp = fopen(infile,"r");
    fgets(buf,sizeof(buf),fp);
    printf("File contents: %s\n",buf);
    fclose(fp);

    xorString(buf);
    fp2 = fopen(outfile,"w");
    fprintf(fp2,"%s",buf);
    fclose(fp2);

    return 0;
}
```

UPDATE:

So I figured out the problem. My xorString() function looks like this:

```
Code void xorString (char *s){
    int i;
    for (i = 0; s[i] != '\0'; i++){
        s[i] ^= XOR_BYTE;
    }
}
```

Which means that when it encounters a 00 byte it stops XORing. How do I in that case make the loop stop at the end of the file without using a 0 byte to do this? I'm at a loss.

hadesgod7979 - September 4, 2019, 7:46 pm

Hello,

corrected code:

```
void xorString (char * s, int nLen)
{
    for ( int i = 0; i < nLen; i ++)
        s[i] ^= XOR_BYTE;
}

int xorFile(char *infile, char *outfile){
    FILE *fp;
    FILE *fp2;
    char buf[1025] = {0};

    fp = fopen(infile,"rb");
    if (fp == NULL) return 0;

    fp2 = fopen(outfile,"wb");
    if (fp2 == NULL) return 0;
    while(!feof(fp))
    {
        int nRead = fread(buf,1, 1024,fp);
        xorString(buf, nRead);
    }
    fclose(fp);
    fclose(fp2);

    return 1;
}
```

myselfandi - September 5, 2019, 8:45 am

(Sep 4, 2019 11:31 AM)pebblak Wrote: [CITE](#) Hey all. I'm really new to C and crypting binaries for that matter so I'm probably missing something obvious.

The issue I'm having is, my file crypter can encrypt and decrypt sample text files with no problem (files containing a line of text), however when tried on a binary, the decrypted binary throws this:

Code ./unencrypted_test_program: line 1: \$\177ELF\002\001\001': command not found
The plain binary is, in fact, 12 KB in size. The encrypted version is 7 bytes, and so is the decrypted version of the encrypted binary. Naturally this doesn't make sense, because I'm using a simple XOR.

Here is my code:

```
Code int xorFile(char *infile, char *outfile){
    FILE *fp;
    FILE *fp2;
    char buf[20000];

    fp = fopen(infile,"r");
    fgets(buf,sizeof(buf),fp);
    printf("File contents: %s\n",buf);
    fclose(fp);
```

```

xorString(buf);
fp2 = fopen(outfile,"w");
fprintf(fp2,"%s",buf);
fclose(fp2);

```

```

return 0;

```

```

}

```

UPDATE:

So I figured out the problem. My xorString() function looks like this:

```

Code void xorString (char *s){
    int i;
    for (i = 0; s[i] != '\0'; i++){
        s[i] ^= XOR_BYTE;
    }
}

```

Which means that when it encounters a 00 byte it stops XORing. How do I in that case make the loop stop at the end of the file without using a 0 byte to do this? I'm at a loss.

You need to get the size of the file first, So something like that would do

```

Code static long GetFileSize(FILE* f)
{
    if(fseek(f, 0, SEEK_END)) // If successful, the function returns zero.
        return 0;

    long fsize = ftell(f); // On failure, -1L is returned,

    if(fsize == -1)
        return 0;

    rewind(f); // void return

    return fsize;
}

```

and your xor function should like like this

```

Code void xorString (char *s, const size_t size){
    int i;
    for (i = 0; i < size; i++){
        s[i] ^= XOR_BYTE;
    }
}

```

Now call your xor function like this

```

Code long fsize = GetFileSize(f);

```

xorString(buf, fsize);also you need to allocate dynamic memory for your file bytes like this

Code char* buf = (char*)malloc(fsize *sizeof(char));since you're manipulating a binary file use "wb" for writing or "rb" reading from a file.

HFDone - September 5, 2019, 11:22 am

you need to read the file bytes to unsigned char pointer

pebblak - September 8, 2019, 11:08 am

Hey, thanks for the responses.

This is my modified code:

```

Code static long GetFileSize(FILE* f)
{
    if(fseek(f, 0, SEEK_END)) // If successful, the function returns zero.
        return 0;

    long fsize = ftell(f); // On failure, -1L is returned,

    if(fsize == -1)
        return 0;

    rewind(f); // void return

    return fsize;
}

```

```

void xorString (char *s, const size_t size){
    int i;
    for (i = 0; i < size; i++){
        s[i] ^= XOR_BYTE;
    }
}

```

```

int xorFile(char *infile, char *outfile){
    FILE *fp;
    FILE *fp2;

    fp = fopen(infile,"rb");
    long fsize = GetFileSize(fp);
    printf("%ld",fsize);
    char* buf = (char*)malloc(fsize *sizeof(char));

    fgets(buf,sizeof(buf),fp);
    printf("File contents: %s\n",buf);

    printf("\n");
    printf("%lu",sizeof(buf));

    xorString(buf, fsize);

    fclose(fp);
    printf("\n");
    printf("%lu",sizeof(buf));

    fp2 = fopen(outfile,"wb");
    fwrite(buf, sizeof(buf), 1, fp2);
    fclose(fp2);

    return 0;
}

```

```

int main(int argc, char *argv[]){

```

```
xorFile(argv[1],argv[2]);
```

```
return 0;
```

```
}
```

The original test binary I'm testing this on is 12KB in size. After running this, the "crypted" version is 4KB. As you can see I added a couple debug statements to see what the size of 'buf' is at two different points. This size prints out as '8'. I don't know what to make of this. The code clearly isn't working. I have no idea why.

jebb - *September 9, 2019, 10:55 am*

```
fp = fopen(infile,"r");
```

change the "r" to "rb". The b stands for binary.

Joseph Ducreux - *October 15, 2019, 5:24 am*

huge difference between text strings and binary data, you will need to read exe or dat byte by byte it into a byte array