Implementing a new Crypto Algorithm – Mysty1

Software Recommended: NetSim Standard v13.0 (32/64-bit), Visual Studio 2017/2019, Wireshark

Project Download Link:

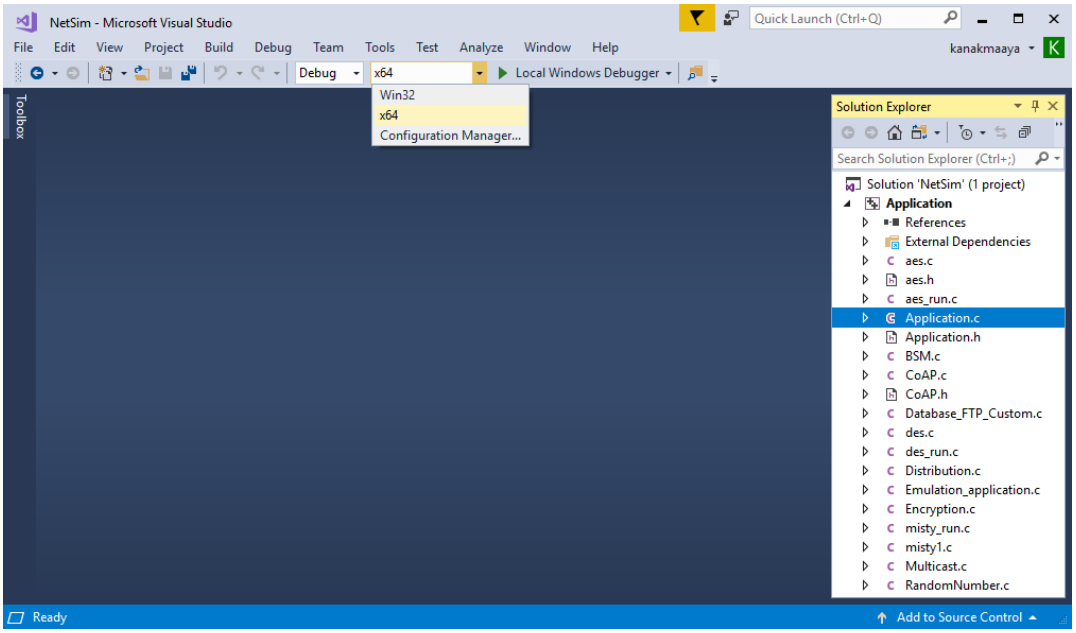
https://github.com/NetSim-TETCOS/MISTY_ENCRYPTION_v13.0/archive/refs/heads/main.zip

Follow the instructions specified in the following link to download and setup the Project in NetSim:

https://support.tetcos.com/en/support/solutions/articles/14000128666-downloading-and-setting-up-netsim-file-exchange-projects

Following modifications were done to the source codes of the Application project:

1. Go to home page, Click on Your work→ Workspace options→ Open code
2. Based on whether you are using NetSim 32 bit or 64 bit setup you can configure Visual studio to build 32 bit or 64 bit Dll files respectively as shown below:



3. Now expand Application Project and click misty_run.c file. This file contains the following lines of code

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "application.h"


void misty_run(char* str, int* len)
{
    int n;
    int l = *len;

    unsigned char buf[32];
    unsigned char key[32];
```

```
for (n = 0; n < *len; n += 16, str += 16, l -= 16)
{
        /* Set the plain-text */
        memcpy(buf, str, min(16, l));

        misty1_main(buf);
        memcpy(str, buf, 16);
}

}
```

In the misty_run() function inside the misty_run.c file we pass the plain text in parts of 16 bytes each time to get it encrypted. This is done because the crypto algorithm accepts a 16 byte plaintext as input. Here the variable str contains the packet payload and len corresponds to the size of payload in bytes.

4. Modifications that were done to the source codes of misty1.c file in the Application project is explained below:

a) Addition of #include<application.h> and #define uint8 unsigned char to the beginning of the misty1.c file(shown in red).

```
#include <stdlib.h>
#include <string.h>
#include "application.h"
 typedef unsigned long u4;
  typedef unsigned char byte;
 #define MISTY1_KEYSIZE 32
#define uint8              unsigned char
```

b) Removed inline keyword that is present before the functions fi(), fo(), fl() and flinv().

```
inline u4 fi( u4  fi_in, u4  fi_key){ ... }

inline u4  fo(u4  *ek, u4  fo_in, byte k){ ... }

inline u4  fl(u4  *ek, u4  fl_in, byte k){ ... }

inline u4  flinv(u4  *ek, u4  fl_in, byte k){ ... }
```

c) Now go to the main() function in the file and check that line #ifdef TESTMAIN was removed or commented before the main() function and also the associated #endif at the end of the main() function.

d) main() function was renamed to unsigned char* misty1_main(uint8* input)

```
unsigned char* misty1_main(uint8* input)
{
/*
   Key:        00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
   Plaintext:  01 23 45 67 89 ab cd ef fe dc ba 98 76 54 32 10
   Ciphertext: 8b 1d a5 f5 6a b3 d0 7c 04 b6 82 40 b1 3b e9 5d
*/


   u4  Key[]= {0x00112233, 0x44556677, 0x8899aabb, 0xccddeeff};
   u4  Plaintext[]= {0x01234567, 0x89abcdef, 0xfedcba98, 0x76543210};
   u4  Ciphertext[]= { 0x8b1da5f5, 0x6ab3d07c, 0x04b68240, 0xb13be95d};
   u4  ek_e[MISTY1_KEYSIZE],  ek_d[MISTY1_KEYSIZE];
   u4  c[4];
```

e) Commented the declaration of Cipher text, Modify the declaration of Plaintext variable, as shown below:

```
u4  Key[]= {0x00112233, 0x44556677, 0x8899aabb, 0xccddeeff};
u4  Plaintext[4];
//u4  Ciphertext[]= { 0x8b1da5f5, 0x6ab3d07c, 0x04b68240, 0xb13be95d};
u4
u4
```

misty1.c

Application (Global Scop

```
283   /* misty1_keyinit(ek_e,Key);
284    misty1_encrypt_block(ek_e,&Plaintext[0],&c[0]);
285    misty1_encrypt_block(ek_e,&Plaintext[2],&c[2]);
286
287    if (!memcmp(c,Ciphertext,4 * sizeof(u4))) {
288      printf("Encryption OK\n");
289    }
290    else {
291      printf("Encryption failed[0x%08lx 0x%08lx 0x%08lx 0x%08lx]\n",
292         c[0],c[1],c[2],c[3]);
293      exit(1);
294    }
295
296    misty1_keyinit(ek_d,Key);
297
298    if (memcmp(ek_e,ek_d,MISTY1_KEYSIZE*sizeof(u4))) {
299      printf("Internal Error keysch is wrong\n");
300      exit(1);
301    }
302
303    misty1_decrypt_block(ek_d,&Ciphertext[0],&c[0]);
304    misty1_decrypt_block(ek_d,&Ciphertext[2],&c[2]);
305
306
307    if (!memcmp(c,Plaintext,4 * sizeof(u4))) {
308      printf("Decryption OK\n");
309    }
310    else {
311
312      printf("Decryption failed[0x%08lx 0x%08lx 0x%08lx 0x%08lx]\n",
313         c[0],c[1],c[2],c[3]);
314      exit(1);
315    }
316    */
```

f) Now check the commented lines starting from misty1_keyinit() to misty1_key_destroy() as shown below:

g) Addition of the following lines of code just above the misty1_key_destroy(ek_e); statement as shown below:

```
// Memcpy is used to equate input which is Char to Plaintext
// which is Unsigned Long

memcpy(Plaintext,input,2*sizeof(u4));
memcpy(&Plaintext[2],&input[8],2*sizeof(u4));
```

```
misty1_keyinit(ek_e,Key);
misty1_encrypt_block(ek_e,Plaintext,&c[0]);
misty1_encrypt_block(ek_e,&Plaintext[2],&c[2])
;

memcpy(input,c,2*sizeof(u4));
memcpy(&input[8],&c[2],2*sizeof(u4));
```

```
// Memcpy is used to equate input which is Char to Plaintext
// which is Unsigned Long

memcpy(Plaintext,input,2*sizeof(u4));
memcpy(&Plaintext[2],&input[8],2*sizeof(u4));

misty1_keyinit(ek_e,Key);
misty1_encrypt_block(ek_e,Plaintext,&c[0]);
misty1_encrypt_block(ek_e,&Plaintext[2],&c[2]);

memcpy(input,c,2*sizeof(u4));
memcpy(&input[8],&c[2],2*sizeof(u4));


misty1_key_destroy(ek_e);
misty1_key_destroy(ek_d);
memset(Key,0,4 * sizeof(u4));
```

h) Inside the misty1_main function the above codes were modified to ensure that the plaintext is properly initialized with the 16 bytes of payload received, for the encryption to happen.

i) Here, memcpy() is done initially to equate input received as which is char, to the plain text which is unsigned long.

```
memcpy(Plaintext,input,2*sizeof(u4));
memcpy(&Plaintext[2],&input[8],2*sizeof(u4));
```

j) After the calls to misty1_encrypt_block() memcpy() is done to equate the encrypted cipher text back to the input.

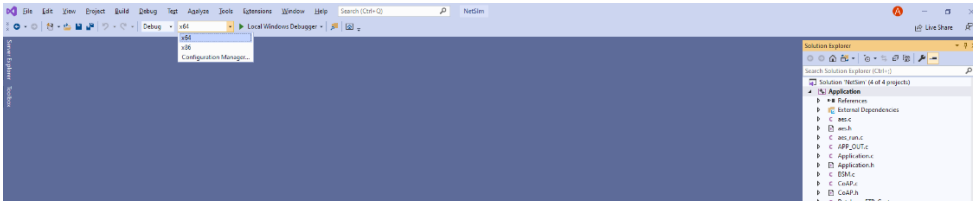
```
memcpy(input,c,2*sizeof(u4));
memcpy(&input[8],&c[2],2*sizeof(u4));
```

5. Now double click on the application.c file and make a call to misty_run() function instead of the call to aes256, inside the copy_payload() function as shown below (changes are marked in red):

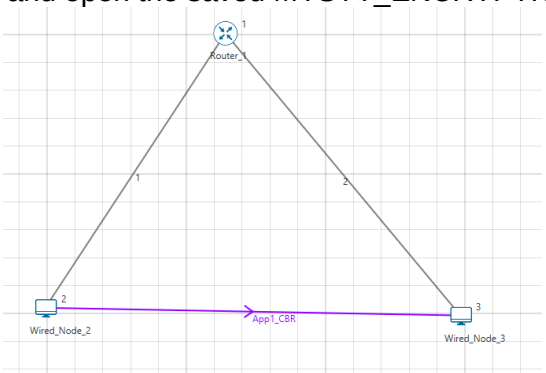
```
if(info->encryption==Encryption_TEA)
                encryptBlock(real,payload,&key);
        else if(info-
        >encryption==Encryption_AES)
        {
                misty_run(real,payload);
                //aes256(real,payload);
```

```
}
else if(info->encryption==Encryption_DES)
        des(real,payload);
```

6. Right click on Solution in Solution Explorer and select rebuild solution
7. Upon rebuilding, libApplication.dll will get created in the bin_x86/ bin_x64 folder.
   Note: While using NetSim 64-bit setup, users need to change solution platform as **x64**



8. Go to Your Work option and open the saved MYSTY_ENCRYPTION_Example.



9. Run Simulation with AES encryption enabled in the Application settings.

10. Now misty1 codes will be running instead of AES256.

11. You can see the encrypted payload in Wireshark either during simulation if online is set or after the simulation if offline is set in the source or destination nodes

12. If Wireshark option is set to offline, then the capture files can be accessed from the results dashboard.