CSED415 Lab 03: Cryptography Report

20230499 / 김재환 / Kim Jaehwan

1. Overview

이번 랩에서는 과제 환경에 접속하여, server.py로 실행되는 서버에서 RC4의 변형 알고리즘인 uRC4를 분석하여 암호화된 키의 원본을 얻어내야 한다. 그리고 얻어낸 키를 이용하여 목표 바이너리 파일로부터 플래그를 얻어내는 것이 최종 목표이다.

2. Phase1: server.py (uRC4)

- uRC4는 RC4 알고리즘의 변형이다. RC4는 스트림 암호로, 초기 유사-랜덤 함수를 초기화한 후 생성되는 난수와 bitwise-XOR 한다. 변경 점은 다음과 같다.

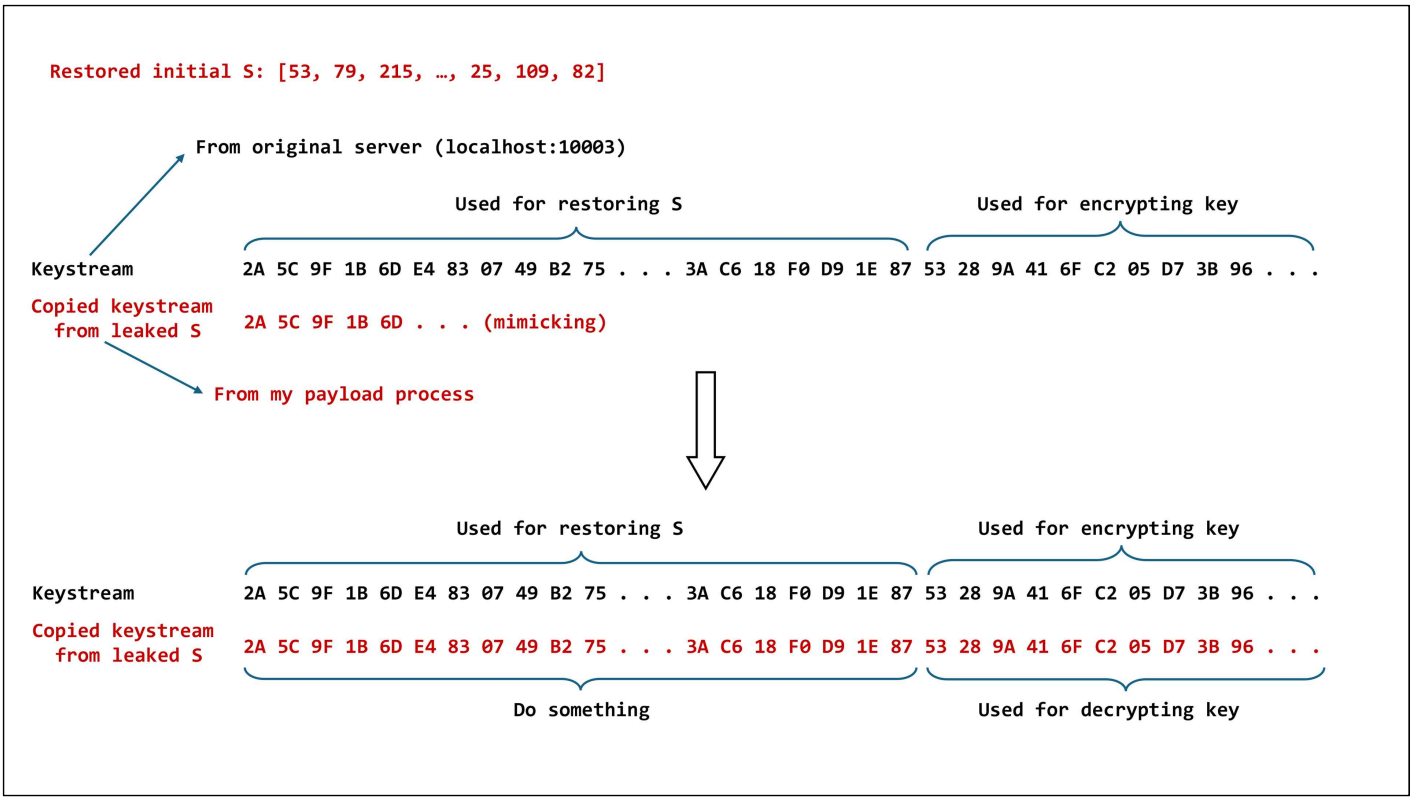
RC4	uRC4

def KSA(key):
  keylength = len(key)
  S = list(range(256))
  j = 0
  for i in range(256):
  j = (j + S[i] + key[i % keylength]) % 256
  S[i], S[j] = S[j], S[i]
  return S
def PRGA(S):
  i = 0
  j = 0
  while True:
  i = (i + 1) % 256
  j = (j + S[i]) % 256
  S[i], S[j] = S[j], S[i]
  K = S[(S[i] + S[j]) % 256] # vanilla RC4
  yield K
def RC4(key):
  S = KSA(key)
  K = PRGA(S)
  return K
	def KSA(key):
	  keylength = len(key)
	  S = list(range(256))
	  j = 0
	  for i in range(256):
	  j = (j + S[i] + key[i % keylength]) % 256
	  S[i], S[j] = S[j], S[i]
	  return S
	def PRGA(S):
	  i = 0
	  j = 0
	  while True:
	  i = (i + 1) % 256
	  j = (j + S[i]) % 256
	  S[i], S[j] = S[j], S[i]
	  K = S[j] # Charlie's modification for uRC4
	  yield K
	def RC4(key):
	  S = KSA(key)
	  K = PRGA(S)
	  return K

- KSA는 초기화 과정, PRGA는 1바이트의 다음 스트림을 생성하는 과정이다.
- 초기 key는 os.urandom() 함수로 초기화되며, 이는 운영체제에서 제공하는 기능을 사용하는 난수 생성 함수로서, 시드를 파악하여 같은 key를 사용하는 것은 불가능하다.
- 그러나 uRC4의 바뀐 로직으로 인해 S를 역추적하는 것이 가능하다. 다음은 KEY_LENGTH = 10일 때 uRC4 알고리즘을 돌리는 과정을 나타낸 것이다.

PRGA 진행 수	현재 상태	비고
초기 상태	idx 0 1 2 3 4 5 6 7 8 9 S 9 8 7 6 5 4 3 2 1 0	S′는 S에서 Swap이 적용된 결과임.
PRGA 1회	i j idx 0 1 2 3 4 5 6 7 8 9 S 9 8 7 6 5 4 3 2 1 0 S′ 9 1 7 6 5 4 3 2 8 0	i = 1, S[i] = 8 j = 8, S[j] = 1 Swap(S[i], S[j]) return S[j] = 8
PRGA 2회	i j idx 0 1 2 3 4 5 6 7 8 9 S 9 1 7 6 5 4 3 2 8 0 S′ 9 1 4 6 5 7 3 2 8 0	i = 2, S[i] = 7 j = (8+7)%10 = 5, S[j] = 4 Swap(S[i], S[j]) return S[j] = 7
PRGA 3회	j i idx 0 1 2 3 4 5 6 7 8 9 S 9 1 4 6 5 7 3 2 8 0 S′ 9 6 4 1 5 7 3 2 8 0	i = 3, S[i] = 6 j = (5+6)%10 = 1, S[j] = 1 Swap(S[i], S[j]) return S[j] = 6
PRGA 4회	i j idx 0 1 2 3 4 5 6 7 8 9 S 9 6 4 1 5 7 3 2 8 0 S′ 9 6 4 1 3 7 5 2 8 0	i = 4, S[i] = 5 j = (1+5)%10 = 6, S[j] = 3 Swap(S[i], S[j]) return S[j] = 5
PRGA 5회	j i idx 0 1 2 3 4 5 6 7 8 9 S 9 6 4 1 3 7 5 2 8 0 S′ 9 6 4 7 3 1 5 2 8 0	i = 5, S[i] = 7 j = (6+7)%10 = 3, S[j] = 1 Swap(S[i], S[j]) return S[j] = 7

- 첫 번째 생성 값을 통해 S[1] = 8임을 얻을 수 있다. 두 번째 값으로는 S[2] = 7, 세 번째에서는 S[3] = 6, 네 번째에서는 S[4] = 5임을 알아낼 수 있다. 다섯 번째 값에서는 S[5]의 값이 아니라, 이전 단계에서 Swap으로 인해 S[3]의 값을 알 수 있다.
- 요점은, RC4 알고리즘에 비해서 **uRC4 알고리즘은 S[j] (S[i], S[j]를 스왑하기 전에는 S[i]) 값을 그대로 반환하기 때문에 리스트 S에 저장된 값을 그대로 알 수 있다**는 것이다. i는 1부터 순서대로 1씩 더해지기 때문에 S[i]의 값을 계속해서 알 수 있다. S[i]와 S[j]를 서로 바꾸는 과정이 있다고 하더라도 **바꾸는 기록을 모두 추적**하면 원래 S의 값을 복원할 수 있다.
- 이를 통해 원본 S의 값을 구하면 PRGA 단계를 처음부터 **시뮬레이션**할 수 있다. uRC4의 알고리즘을 그대로 작성하면 원본 S를 통해 **keystream을 복제**할 수 있고, 이를 통해 원하는 부분의 스트림을 이용하면 복호화에 사용할 수 있다.
- 원본 S를 계산하는 과정에서 일정 길이의 스트림을 사용했으므로, 복제한 키 스트림의 앞부분을 그 길이만큼 사용한 이후 생성 값을 사용하여 암호화된 키에 XOR하여 복호화에 사용한다. 다음과 같이 키 스트림을 복제한 후 **키 암호화에 사용된 키 스트림과 동일한 부분을 다시 사용**하여 키를 복호화할 수 있다.



3. Phase2: target (2048)

- 2048 게임에서 매우 높은 점수인 3,932,156보다 높은 점수를 얻으면 플래그를 얻을 수 있다. 그러나 2048 게임에서 숫자 n이 적힌 블록 두 개를 합쳤을 때 얻는 점수는 $2n$이므로 해당 점수를 얻는 것은 매우 어렵다.
- main.c 코드를 분석해 보면, 점수를 저장하는 변수의 자료형이 short 형이므로 최대 65,535까지만 저장할 수 있다. 따라서 엄청난 실력과 운으로 게임을 하더라도 정상적인 방법으로 플래그를 얻는 것은 불가능하다.
- ALSR, NX, Canary 등의 보안 기법이 적용되어 있으므로 BOF 등의 방법을 사용하는 것은 어렵다.
- 그러나 **점수를 비교하는 로직을 보면 unsigned 형과 비교한다. 따라서 점수가 음수가 되는 경우 해당 조건을 만족**시킬 수 있다.

```
// Max score: https://en.wikipedia.org/wiki/2048_(video_game)
if (score > 3932156U) {
        printf("Congratulations! You've scored %lu points!\n", score);
        print_flag("2048");
        exit(0);
}
```

- 또한, 다음과 같이 **정해진 입력이 아닌 경우 점수에 1을 빼는** 것을 알 수 있다.

```
switch(c) {
        // omitted
        default: {
                char msg[100];
                snprintf(msg, sizeof(msg), "> unknown cmd: %s (-1 point)\n", input);
                printf("%s", msg);
                score--;
                success = false;
        }
}
```

- 초기 점수는 100부터 시작하므로 그보다 더 많은 잘못된 입력을 넣어주면 현재 점수를 음수로 만들 수 있다. 점수가 음수가 되면 unsigned 자료형과 비교를 통해 3,932,156보다 커야한다는 조건을 만족시켜 플래그를 얻을 수 있다. 이후 아무 입력을 통해 보드가 변화하도록 하면 되는데, 이때 한 방향의 입력만 사용 시 초기 상태에 따라서 보드가 변화하지 않을 수 있으므로 나의 경우에 w와 s를 둘다 넣어 플래그를 얻어내었다.

```
for _ in range(200):
        p.send(b"1") // 1은 잘못된 입력이므로 반복문 이후에 score는 -100이 된다.
```

- 해당 취약점을 수정하기 위해서는 코드 구현의 의도에 따라 두 가지 방법으로 수정할 수 있다.
- 첫 번째는 **점수가 음수가 되지 않도록 구현**하는 것이다. 잘못된 입력에 대해 점수를 깎되, 최저 점수는 0점으로 제한한다. 잘못된 입력을 받았을 때 현재 **점수가 0이라면 점수를 그대로 둔다.**

```
switch(c) {
        // omitted
        default: {
                char msg[100];
                snprintf(msg, sizeof(msg), "& unknown cmd: %s (-1 point)\n", input);
                printf("%s", msg);
                if (score > 0) score--; // fixed, adding 'if'
                success = false;
        }
}
```

- 두 번째 방법은 **점수가 음수가 되는 것이 가능하도록 구현**하는 것이다. 이 경우에는 성공인지 비교할 때 **부호 있는 자료형과 비교**해야 한다. 물론 이 경우 score 변수가 short 자료형이어서 조건을 달성할 수 없게 되므로 절대 플래그를 얻을 수 없다.

```
// Max score: https://en.wikipedia.org/wiki/2048_(video_game)
if (score > 3932156) { // fixed, removing 'U'
        printf("Congratulations! You've scored %lu points!\n", score);
        print_flag("2048");
        exit(0);
}
```

4. Result

exploit.py를 통해 성공적으로 키와 플래그를 얻어내었다. 기록을 첨부하였다.

```
csed415-lab03@csed415:/tmp/asdfdir$ python3 exploit.py
[~] Generating process ...
[~] Connected to the target: server.py.
[~] Restoring original S ...
[!] Original S is restored:
[53, 79, 215, 99, 48, 210, 233, 154, 193, 107, 101, 152, 34, 218, 7, 134, 93, 235, 241, 6, 84, 183, 32, 216, 59, 184,
77, 49, 27, 61, 138, 2, 199, 62, 198, 222, 3, 147, 86, 56, 52, 180, 63, 203, 19, 92, 187, 64, 106, 42, 118, 97, 221,
122, 58, 44, 248, 162, 80, 156, 195, 192, 250, 72, 197, 150, 1, 88, 74, 164, 78, 16, 155, 253, 234, 30, 139, 137, 70,
46, 136, 212, 243, 168, 98, 40, 135, 67, 141, 176, 115, 232, 73, 100, 123, 5, 153, 244, 37, 146, 144, 166, 140, 172, 22,
38, 174, 36, 112, 213, 245, 4, 114, 224, 246, 116, 119, 167, 175, 76, 51, 108, 171, 31, 12, 157, 131, 158, 45, 117, 15,
207, 237, 182, 160, 13, 190, 177, 103, 196, 240, 65, 143, 68, 163, 128, 185, 169, 151, 26, 249, 130, 239, 225, 209, 194,
191, 204, 54, 188, 201, 28, 173, 205, 39, 186, 145, 127, 254, 178, 149, 214, 47, 247, 219, 35, 179, 50, 181, 110, 14,
75, 231, 148, 81, 96, 251, 66, 223, 200, 87, 208, 23, 217, 33, 227, 126, 125, 242, 71, 236, 226, 211, 20, 24, 230, 220,
90, 94, 8, 89, 229, 113, 202, 252, 60, 0, 83, 142, 95, 104, 55, 228, 9, 133, 57, 165, 18, 43, 105, 124, 161, 129, 120,
10, 255, 85, 102, 91, 238, 21, 159, 170, 132, 189, 111, 206, 17, 69, 121, 29, 41, 11, 25, 109, 82]
[~] Generating keystream ...
[~] Mimicking ...
[~] Encrypted key:
b'D50FD313392A41408F6C00320341A62FE50B70C91E8CB21FAF81C3B86CDD52D8FF6494FCE4DBF1847EBD9B30C29B311D0160E22B88F9EC193DB
F33B7D60C79FEB1\n'
[@] Decrypting :
D50FD313392A41408F6C00320341A62FE50B70C91E8CB21FAF81C3B86CDD52D8FF6494FCE4DBF1847EBD9B30C29B311D0160E22B88F9EC193DBF3
3B7D60C79FEB1 ...
[@] Decrypting done! : e933a52be418ef6c160152d4a7581efeee51eb25ae2452cd0deec507cc957a65

[!] Decrypted key: e933a52be418ef6c160152d4a7581efeee51eb25ae2452cd0deec507cc957a65

[~] Closing the connection: server.py.

[~] Generating process ...
[~] Connected to the target: target.
[~] Making score be negative ...
[!] Flag found ...
944583A6CFFB89C892AEABE82B57E278D32A28E60DFA74D4ABFFE512C9F415F1
5A0B9A14013B8C46FC58118F3CEC895C116F5F1450D8D153F587D6FA4A5AEB76
12B7478E8F97156A8A1FB474D4CEA024DD9E437B5F1E3744418EFC9CF383DDEC
197A1E984587655A28ECCA91EB8949BE308544A0DA687A15C17D443A9D008C70
35E2B030BA4BD41035D656139EC279C9CE7E33E88204C7C76EBA29F6B66F6AA7
C51506CA2EC0E2E8108D1C066A9CF61674DB2F2E9C7AFFADA04F57C3E3742726
9BB98C60D27E0F9B4CAB2307019DBC418B3CFEB85A3342216719F6E92B8C047B
248BF943BC6B2B1166E7AEC84F9CEC1C32B095F6B4FED89F9C2184151150B47B
B9C44B53CBEF17BE34E80657C58F0E7F0AB45240CED63C05F24761D11C90E1DC
7F02635A24351E2F14C2A8F0C77E6E3753862547DD19F17ADBD8CB7970831B01
4CA035156F107F3E7C80BB11DBFA850747444869911CBCF450A3C9ADE93CE3AF
FBB4B5155D7BB09908874B78AAE50DFDD0D81045DD7EB9A0C3D0B9AB6895CE77
CC57665043150EEC45318B3C7332E331A73FA4FFC2AF1913FEDB367830C73857
960B1E921BEEA9712B38D3D201DB616319F1059FB56E9857769D55F3B9AFED9F
1F89CFFD327A490CF033D48C95F228FBF41DF1D46BD0F8246C607362FBB02336
2AC1EFF712DF4B1A92A8E4BF787143B31068D8DFA6D9BEF769ECD0BB7E58673D
[~] Closing the connection: target.
```