Hugh Lavery – 14313812

3D5A – Lab 1 Hash Tables

Task 1

For my linear probing hash function I just went with a basic (strlen(key)*t)%table size. Where t is the ascii value of the first character in the input array to be hashed. If that index of the hash table was found to be full then my function will just check the next slot in the table (looping back to the start) until it finds an empty space. With an input array of 9 keys this only required 2 probes for one key and three for another while the rest could enter the table in the first probe.

For my double hashing function the initial hash is the same as the linear probing hash. Then instead of just probing linearly I used another hashing function which is t*p % (table_size-1) + 1;. Where p is a large prime number 233. This is then added to the initial index found by the first hash function and again modulus to the table size. The load calculated for my table is 0.471, which means that it can still take more entries before I would need to consider needing a bigger hash table. The coverage is good, when I tested my code by attempting to put more char arrays into the table it correctly exits the function and tells me the table is full and the load is 1.

Task 2

The open source project whose hash function I decided to review is OSSEC by Trend Micro Inc. Their project can be viewed on githib [1]. OSSEC is described as a full platform to monitor and control your system. It performs host-based intrusion detection, log monitoring and security information and event management in one powerful solution.

One of the types of hashing functions used in this project is MD5 hashing algorithm. This algorithm was originally made to be used as a cryptic hash function but was found to be quite vulnerable. From my interpretation, in this project it is used as a checksum to verify data integrity. Something that would be very useful for a platform which is looking for intrusion detection.

The main parts of the MD5 algorithm is the step function which adds 16 longwords of data and then rotates the bits.

```
#define MD5STEP(f, w, x, y, z, data, s) \
( w += f(x, y, z) + data,  w = w<<s | w>>(32-s),  w += x )
```

Which is used by the core of the algorithm

```
void MD5Transform(uint32 buf[4], uint32 const in[16])
{
    register uint32 a, b, c, d;

    a = buf[0];
    b = buf[1];
    c = buf[2];
    d = buf[3];

    MD5STEP(F1, a, b, c, d, in[0] + 0xd76aa478, 7);
```

This MD5STEP is then repeated 16x4 times. 16 times for each of F1-F4 which are the four core functions

```
#define F1(x, y, z) (z ^ (x & (y ^ z)))
#define F2(x, y, z) F1(z, x, y)
#define F3(x, y, z) (x ^ y ^ z)
#define F4(x, y, z) (y ^ (x | ~z))
```

[2]

This hash function differs a lot from ones I have seen up to this point. It involves a lot of bit manipulation and is more complex. I also was unaware until not that hash functions can serve much more purose then just filling a hash table or cyriptically storing data. Here it is used to ensure data stored hasn't been manipulated by some malware. This type of hash is know as a message digest

algorithm. This means it will prompt the owner to notify them to any change made to data

References

[1] – https://github.com/ossec/ossec-hids - Trend Micro Inc
[2] - https://github.com/ossec/ossec-hids/blob/master/src/os_crypto/md5/md5.c Author Colin Plum (1993)