Scott Christensen

09/14/16

C.S. 465

Hash Attack Technical Report

For my implementation of this Lab, I first studied the topic materials given to us in class and then began my project using the provided code. The first of the main functions I implemented was associated with collision attacks. To do so, it encrypted and truncated a given string and see if that value matched any if a Hash Map. If not, I would save it into that Hash Map and repeat the process till a match was found, therewith saving the number of times we had to loop through this process. This process I had repeat a number of times in order to get a good average number of times it took for any given n (where n is number of bits used as input).

What I found was that the number of times it took on average to find a matching hash was always a little bit more than what the expected values were. However, this difference was relatively small across every value, which led me to believe that there was a method in my code that took k\*n time to complete (where k is some constant). In this case, I assumed that my random string generator was to blame. Other than this, it modeled the expected values quite closely.

Next, I implemented the function associated with second pre-image attacks. To do so, it used the same idea and functionality as the collision attacks implementation, but this function would not save any string other than the first. Each subsequent string that was encrypted and truncated was compared only against the first. Upon running the function with different values of n, I found that the results were very much smaller than the expected values. This result I can only assume occurred due to outliers in the data and comparing against the worst case scenarios derived from the expected values.

In conclusion, I feel that I learned a lot during the implementation of this project. I feel that I’ve gained a solid understanding of the algorithms used and, from comparing the results, realize just how hard a hash attack hacking attempt would be in the real world. Glad to know that hashing is a secure practice.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Hash Attack Lab Results |  |  |  |  |  |  |  |
| Results showing average number of iterations to find a match | | | | |  |  |  |
|  |  |  |  |  |  |  |  |
| Number of bits | 4 | 8 | 10 | 12 | 14 | 16 | 18 |
| Expected Collision Test | 4 | 16 | / | 64 | / | 256 | / |
| Collision Test | 5 | 21 | / | 77 | / | 319 | / |
| Expected Pre-Image Attack | 16 | 256 | 1024 | 4096 | 16384 | 65536 | 262144 |
| Pre-Image Attack | 16 | 245 | 271 | 3928 | 7013 | 36902 | 82195 |
|  |  |  |  |  |  |  |  |
|  | 20 | 22 | 24 | 28 | 32 | 36 | 40 |
|  | 1024 | / | 4096 | / | 65536 | 262144 | 1048576 |
|  | 1360 | / | 19666 | / | 85001 | 402021 | 1101942 |
|  | 1048576 | 4194304 | 16777216 | / | / | / | / |
|  | 760626 | 994267 | 8930441 | / | / | / | / |

**package** Project2;

**import** java.util.\*;

**public** **class** HashAttack

{

**final** **protected** **static** **char**[] ***hexArray*** = "0123456789ABCDEF".toCharArray();

**public** **static** **void** main(String[] args) **throws** Exception

{

//Do test for bit size = 4

//preImageTestWrapper(4, 100);

//Do test for bit size = 8

//preImageTestWrapper(8, 100);

//Do test for bit size = 10

//preImageTestWrapper(10, 100);

//Do test for bit size = 12

//preImageTestWrapper(12, 100);

//Do test for bit size = 14

//preImageTestWrapper(14, 10);

//Do test for bit size = 16

*preImageTestWrapper*(16, 10);

//Do test for bit size = 18

//preImageTestWrapper(18, 10);

//Do test for bit size = 20

//preImageTestWrapper(20, 10);

//Do test for bit size = 22

//preImageTestWrapper(22, 10);

//Do test for bit size = 24

//preImageTestWrapper(24, 3);

//Do test for bit size = 4

//collisionTestWrapper(4, 100);

//Do test for bit size = 8

//collisionTestWrapper(8, 100);

//Do test for bit size = 12

//collisionTestWrapper(12, 100);

//Do test for bit size = 16

//collisionTestWrapper(16, 100);

//Do test for bit size = 20

//collisionTestWrapper(20, 100);

//Do test for bit size = 24

//collisionTestWrapper(24, 100);

//Do test for bit size = 28

//collisionTestWrapper(28, 100);

//Do test for bit size = 32

//collisionTestWrapper(32, 5);

//Do test for bit size = 36

//collisionTestWrapper(36, 5);

//Do test for bit size = 40

//collisionTestWrapper(40, 5);

}

//provided encryption code

**public** **static** **byte**[] encrypt(String x) **throws** Exception

{

java.security.MessageDigest d = **null**;

d = java.security.MessageDigest.*getInstance*("SHA-1");

d.reset();

d.update(x.getBytes());

**return** d.digest();

}

//gets an average

**public** **static** **int** getAverage(List<Integer> input)

{

**int** result = 0;

**for**(Iterator<Integer> i = input.iterator(); i.hasNext();)

{

result += i.next();

}

**return** result / input.size();

}

//wrapper for preImage

**public** **static** **void** preImageTestWrapper(**int** bitSize, **int** numRuns) **throws** Exception

{

List<Integer> runs = **new** ArrayList<>();

**for**(**int** i = 0; i < numRuns; i++)

{

runs.add(*preImageAttack*(bitSize));

}

**int** output = *getAverage*(runs);

System.***out***.println("For bit size = " + bitSize + " with " + numRuns + " runs: " + output);

}

//wrapper for collisionTest

**public** **static** **void** collisionTestWrapper(**int** bitSize, **int** numRuns) **throws** Exception

{

List<Integer> runs = **new** ArrayList<>();

**for**(**int** i = 0; i < numRuns; i++)

{

runs.add(*collisionAttack*(bitSize));

}

**int** output = *getAverage*(runs);

System.***out***.println("For bit size = " + bitSize + " with " + numRuns + " runs: " + output);

}

//returns number of iterations to find a collision against a certain string for the number of bits given (needs to be a multiple of 4!!!)

**public** **static** **int** preImageAttack(**int** numBits) **throws** Exception

{

RandomString randomStringGen = **new** RandomString(10);

**int** count = 0;

HashMap<String, String> hashes = **new** HashMap<>();

**boolean** found = **false**;

**boolean** putIn= **false**;

**while**(!found)

{

String str = randomStringGen.nextString();

String truncHash = *bytesToHex*(*encrypt*(str)).substring(0, numBits / 4);

**if**(hashes.get(truncHash) != **null**)

{

found = **true**;

//System.out.println("Found collision with inputs: " + str + " & " + hashes.get(truncHash) + " at hash: " + truncHash);

}

**else** **if**(putIn == **false**)

{

hashes.put(truncHash, str);

putIn = **true**;

}

count++;

}

**return** count;

}

//returns number of iterations to find a collision for a hash of the number of bits given (needs to be a multiple of 4!!!)

**public** **static** **int** collisionAttack(**int** numBits) **throws** Exception

{

RandomString randomStringGen = **new** RandomString(10);

**int** count = 0;

HashMap<String, String> hashes = **new** HashMap<>();

**boolean** found = **false**;

**while**(!found)

{

String str = randomStringGen.nextString();

String truncHash = *bytesToHex*(*encrypt*(str)).substring(0, numBits / 4);

**if**(hashes.get(truncHash) != **null**)

{

found = **true**;

//System.out.println("Found collision with inputs: " + str + " & " + hashes.get(truncHash) + " at hash: " + truncHash);

}

**else** hashes.put(truncHash, str);

count++;

}

**return** count;

}

**public** **static** String bytesToHex(**byte**[] bytes)

{

**char**[] hexChars = **new** **char**[bytes.length \* 2];

**for** ( **int** j = 0; j < bytes.length; j++ )

{

**int** pos = j \* 2;

**int** v = bytes[j] & 0xFF;//AND them with 0xFF, then shift 4

hexChars[pos] = ***hexArray***[v >>> 4];

hexChars[pos + 1] = ***hexArray***[v & 0x0F];

}

**return** **new** String(hexChars);

}

}

**package** Project2;

**import** java.util.Random;

**public** **class** RandomString {

**private** **static** **final** **char**[] ***symbols***;

**private** **final** Random random = **new** Random();//random number

**private** **final** **char**[] buf;//our buffer

**static** {

StringBuilder tmp = **new** StringBuilder();

**for** (**char** ch = '0'; ch <= '9'; ++ch)

tmp.append(ch);

**for** (**char** ch = 'a'; ch <= 'z'; ++ch)

tmp.append(ch);

***symbols*** = tmp.toString().toCharArray();

}

**public** String nextString()

{

**for** (**int** idx = 0; idx < buf.length; ++idx)//fill our predefined buffer with a random symbol, which

//will be our string

{

buf[idx] = ***symbols***[random.nextInt(***symbols***.length)];

}

**return** **new** String(buf);

}

**public** RandomString(**int** length)

{

**if** (length < 1)

**throw** **new** IllegalArgumentException("length < 1: " + length);

buf = **new** **char**[length];

}

}