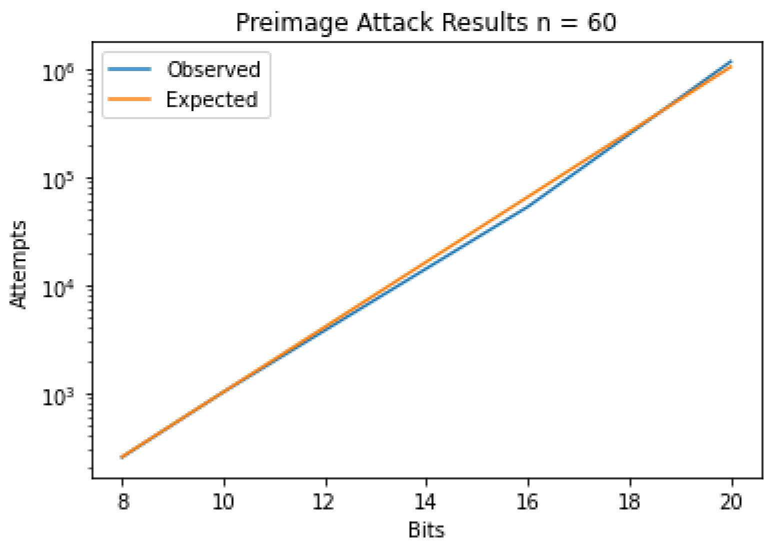
Hash Attacks

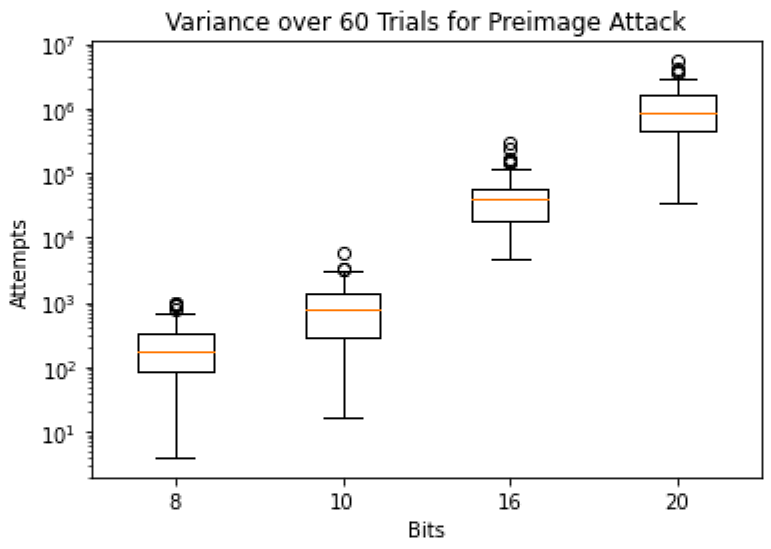
Written by Chase Hiatt, Reviewed by Nick Pixton

In this lab, we were tasked with recreating two real world attacks used on hashing systems. The first technique we used is known as a ‘preimage attack’. This type of attack is done when a user attempts to find a message that has the same hash as a specified target message. For example, imagine a hacker has acquired a user’s hashed password from a recent database leak. Due to the one-way nature of hashing, the attacker cannot directly find the plaintext that produces the given hash. Therefore, to extract the plaintext from a hashed password, the hacker may try to do a preimage attack. To execute this type of attack, the hacker essentially has to run a bunch of values through the hash function until they find the value which produces a hash matching the original desired value.

If we assume the hash function produces a digest of size n, then this type of reversing is expected to take 2^n attempts on average to find a match. This expectation is derived from the presumed randomness of the hash function. If we assume the use of a hash with a fairly random and uniform distribution, then the expected time to find a matching hash is the expected time of finding a specific value in a distribution with a range of 2^n. Obviously finding a specific value from a range of 2^n takes 2^n attempts on average, which gives us our expected run time. While searching through this many values is feasible for smaller sizes of n (i.e. 10,20,25), it becomes intractable as n grows. Modern hashing algorithms such as SHA-3, with output sizes as large as 512 bits, make this type of attack completely infeasible in the general case. It is important to note that while the general case of this attack is very difficult, it can be made easier for certain choices of message. For example, if you hash the message ‘password’, a hacker will likely be able to use a dictionary attack to reverse this easily. For this reason, it is important to salt all passwords prior to hashing them.

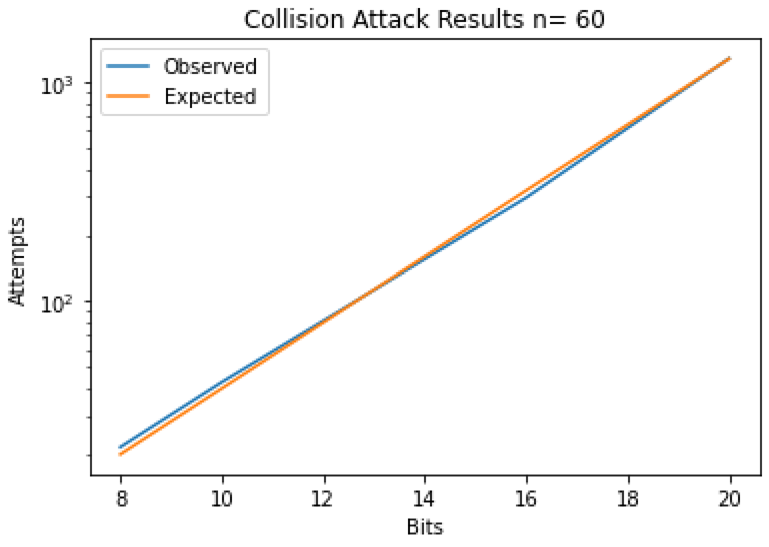
For this lab, we used a SHA-1 wrapper that allows us to easily truncate SHA-1’s output to a predetermined size. By using only certain portions of SHA-1, we were able to attack a real hashing algorithm without having to iterate through 2^160 values to find a collision. For my tests, I chose to run preimage attacks on bit sizes of 8,10,16, and 20. For each of these sizes, I ran 60 trials and graphed the average number of attempts until a match was found. In the figure below, you can see my observed and expected results graphed. With my sample size of 60, my observed and expected results match up closely.

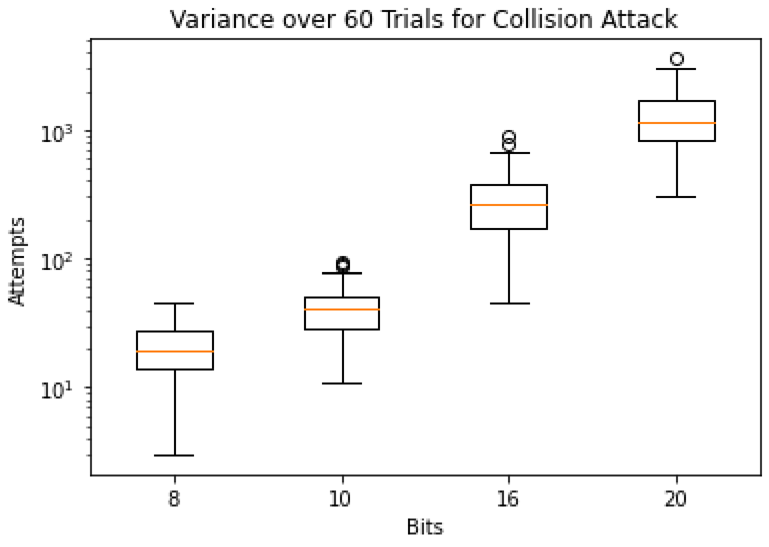




After exploring preimage attacks, we moved on to another exploitation of hash functions, the birthday attack. This attack, also known as a collision attack, happens when an attacker tries to find two arbitrary inputs that hash to the same value. This type of attack can be executed far quicker than a preimage attack as a result of the exponential searching power of the ‘any pair equal’ relationship. To better understand this, consider the number of potential matches after hashing the nth value in a sequence. In the preimage attack, all hashes only had 1 potential match. Each hash would either match the target, or it would not. This means there is a linear exploration of the search space. In the case of a birthday attack, when you hash the nth value, it can potentially match with n-1 other values. If hash 2^10 values, you have actually checked 2^20 potential pairs.

This exponential searching allows us to derive the expected runtime of a birthday attack as being O(2^(n/2)). After running attacks on bit sizes of 8,10,16, and 20, my experimental results matched this trend. In my particular case, the leading constant which most closely matched my real world results was ~ 1.25. With 60 trials of each of the 4 bit sizes, I am confident my code behaves as expected. Below is a chart comparing the expected number of attempts to find a collision vs actual average number of attempts. Additionally, I graphed the variance in attempts for the different bit sizes.


Collision Attack Results n= 60
Observed
Expected
Attempts
10^3
10^2
Bits


Variance over 60 Trials for Collision Attack
Attempts
10^3
10^2
10^1
Bits