Matthew Kearney
Professor Tegan Brennan
Security Privacy and Society
23 April 2024

Homework 3

## Problem 1: Password Cracking! (30%)

Hooray! You have compromised an online server and stolen their list of stored passwords along with the associated usernames. Time to break into those accounts. One problem though: once you open the password file, you discover that only the password *hashes* were stored. Guess more security controls were in place than you thought. Let's see if we can recover any passwords anyway!

**(1)** You figure that some passwords are more common than others. Possibly there are some users who chose very common passwords. And luckily for you, there are many password lists available. The RockYou wordlist is one such password list, consisting of passwords leaked in a data breach. The entire wordlist is huge; over 14 million unique passwords. Let's start by just downloading the top 2000 words. You can find them in the file rockyou2000.txt under the Canvas course module labelled Password Cracking Code.

**(a)** You don't know what hash function the target passwords were hashed with, but you're guessing it's a common one. First, hash every password of rockyou2000.txt using md5 (an insecure hash, but perhaps the server used it anyway). Submit the results in a .txt file rockyou2000md5.txt in the form of **plaintext password: resulting hash in hex format** with one such pair per line.

**Also submit the code or script you wrote to do this**. Note: you don't need to write the md5 hash function itself; many popular programming languages have their own built-in cryptographic libraries that can do this for you (e.g. hashlib in Python).

**(b)** Do the same thing using the sha256 hash function. Again submit both the resulting file rockyou2000sha256.txt and the corresponding code or script.

**(2)** Time to see if any of our precomputed hashes are a match! Download the file of leaked password hashes: leakedhashes.txt in the Canvas module. Check if any of your precomputed hashes matches the hashes of the users. Did you get any hits? Submit a file listing the **user: plaintext password** pairs of all passwords you cracked using this method (note: it will not be all passwords).

**(3)** What security controls could have been implemented to help mitigate this attack? Name and describe the potential mitigation impact of at least two distinct controls.

**(4)** There are many password cracking tools available to level up our skills.Take a look at Hashcat here and read this article. Then list and describe the different kinds of attacks given in the article.

(1) implemented hashing on list of the top 2000 words in md5.py and sha256.py
(2) main.py compiles (checks for matching hashes) from the sha256 and md5 hashing functions with the leaked hashes observed. At first I didn't find matches as I was incorrectly searching for matches. After correcting this, I found 13 (distinct) matches and stored them in 'finds.txt', where we start with user0 (i=0) and go up to user49.
(3) Password salting is the first method that could be used to mitigate (the success) of attacks. If an attacker were to be successful at finding user matches, then if the storage

from where the leaked hashes originated had salted the passwords then the attacker would have to check all password-salt combinations which is computationally infeasible.

Also, potentially a more simple and logistical way to mitigate this attack is by increasing the requirements for allowed passwords in the first place. As we can see redundantly from the top 2000 list, many of these words are single names, ideas, characters, cities, countries, etc all in lowercase. Maybe they sometimes have a 1 etc at the end, but besides that they crucially lack variety and deeper levels of chaos that is only possible to humans. This is the necessity that most of us can realize in elaborated modern passwords and turning to other authentication schemes. If the system had requirements for passwords being more than (say) 10 letters, and that they have to have numbers, special characters, capitalized characters, etc, then none of the users from the observed (leaked) hashes would have been compromised.

(4) The article serves the concept of a rule-based attack well by demonstrating Hashcat's functionality to employ a set rulings on a set of words as a means of getting access to a greater variety of possible passwords. A contextual attack may rely on certain information about a particular entity. Combinator attacks seem to use multiple wordlists and rules to create a hefty list of candidate passwords. Topographical attacks look at the overarching patterns in all of the passwords (and try to notice anomalies). Markov attacks common patterns in passwords through their distribution. Dynamic attacks are the potential conjunction or disjunction of all of these attacks, as attacks are generally composed of multiple smaller vulnerabilities and are hacked through iteration and adaptation.

## Problem 2: Understanding NSEC5 (35%)

Introduced by Goldberg *et al.* and currently under IETF's consideration for standardization, *NSEC5* comprises a replacement of the NSEC3 protocol for removing an identified vulnerability. Read the Introduction section from the original paper, and answer the following questions.

**(1)** What *type of leakage* does NSEC3 allow for and what is the *technical reason* behind such vulnerability?

**(2)** What are the core *technical features* of the new cryptographic hash function used by NSEC5 that allows for correct verification of non-existing domain names, yet preventing the type of leakage NSEC3 allows?

**(3)** NSEC5 requires distributing a *signing key* to any secondary DNS resolver, who by definition is assumed to be *untrusted* with respect to validity of the provided answers for non-existing domain names. How is this technical choice justified?

**(4)** In terms of *secure system design*, what is the lesson learnt in view of what justified the development of protocols DNSSEC, NSEC, NSEC3 and NSEC5?

(1) NSEC3 has a zone enumeration problem which allows attackers to learn IP addresses of hosts in a zone, which essentially means they can walk through the host architecture and learn about their server and application that they should not have access to. This 'leaked' information through the zone enumeration vulnerability is a result of not

employing public key crypto in the online protocol on the DNSSEC nameserver. Specifically, without such online protocol, the researchers proved that DNSSEC with NSEC and NSEC3 records cannot simultaneously stop attackers from tampering with DNS messages and maintain privacy against zone enumeration bringing the new cryptographic construction with online public-key crypto that fully protects from zone enumeration.

(2) The core technicality that removes the zone enumeration vulnerability from NSEC3 in NSEC5 is the replacement of the unkeyed hash function with a deterministic RSA-based keyed hashing scheme. NSEC5 relies on RSA keys for its operation so it's not susceptible to the same types of attacks as the unkeyed hash function used in NSEC3. Even if the secret NSEC-5 hashing key is compromised, zones remain protected against network attackers and infected DNS. However, NSEC5 security is relaxed to the level of NSEC3 as it becomes vulnerable to the same zone enumeration suite of attacks upon the key leaking.

(3) The understanding was made that the premise of soundness where we trust the primary name server and privacy where both primary and secondary servers are expected to keep the contents of R confidential are not solvable together. As our privacy condition gives trust to the secondaries, NSEC5 introduces a new aptly named 'secondary' secret key. NSEC5 uses RSA-construction of $PK_s = (N_s\ e_s)$ and $SK_s = (N_s\ d_s)$ to maintain trust across zones as in the primary server properly designating correct IP addresses for domain names (and to not malfunction or create, allow harm) and for the primary and secondary trust to only reveal information ("not existent") about the particular query. We still only allow the primary to have the primary secret key. However, even if the secondary key is leaked, NSEC5 is still resistant to zone enumeration through precomputed denial of existence.

(4) While all four protocols are meant to protect from vulnerabilities in DNS architecture, it is important to note that none of these protocols warrant or claim perfect security. It is true that as vulnerabilities are discovered, countermeasures are developed to stop future attacks of the vulnerability. However, there is a large sea of vulnerabilities in our computing systems and so the lesson learned is that we should not expect some final model of DNSSEC's evolution to someday attain perfect secrecy as new vulnerabilities to DNS infrastructure and even issues within the actual security extension are always discovered.

## Problem 3: Intrusion detection (35%)

Introduced by Juels and Rivest, *Honeywords* comprise a non-cryptographic method for hardening password security against stolen password files after successful breaches into authentication servers. The idea is to employ *decoy* passwords so that any user's account is associated with not only one (the real) password, but also with many fake ones, and then distribute password verification *across two servers*, say one red and one blue, each storing and verifying "half" of the credentials needed to be verified in order to successfully authenticate a user (see also Figure 1). Read about the Honeywords authentication system from the original paper, and answer the following questions.

**(1)** How does this *split architecture* improve security? Consider the two cases where an attacker compromises only one of the servers.

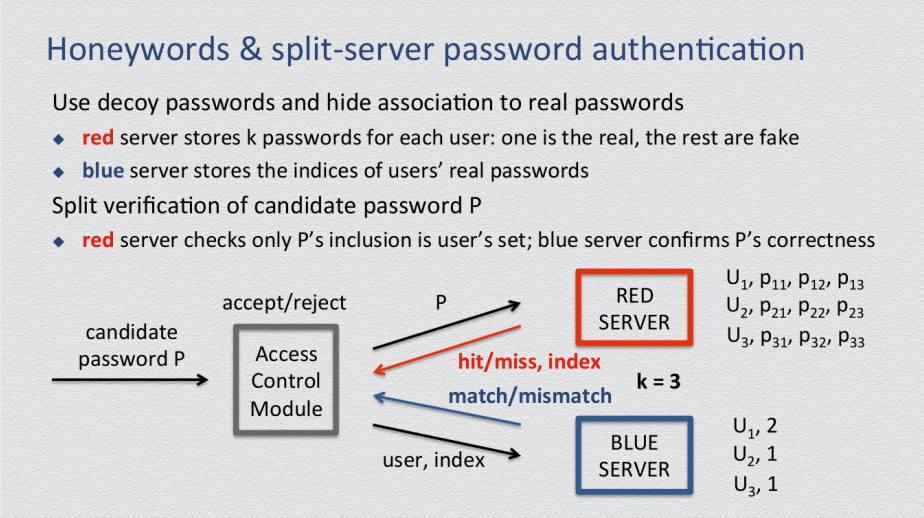


Figure 1: Hardening password security by employing decoy passwords in a split-server architecture.

**(2)** How does this system make password cracking *detectable*.

**(3)** What constitutes a *good* honeyword for a user whose real password is `Bo$tonRedSox76`, if honeywords are generated by tweaking real passwords? Provide a few examples.

**(4)** Stevens employs the Honeywords authentication system, and you just stole the passwords list, `00000000000000000000000000`, `itWb!%s45_3gMoI00286!*mooewTi409##21jUi`, `7304ASpaceOdyssey`, and `2001ASpaceOdyssey`, of an employee in the Office of the Registrar. If you have *only one* chance to impersonate them (and try to increase your GPA), which password will you choose and why?

Unfortunately the paper would not be viewed for me but I researched on my own.

(1) This model improves security as if an attacker has access to the users and their real and fake associated passwords, they will not know which are real and which are fake and further attack techniques would need to be employed. However with this list the attacker does have access to brute force the passwords of the user(s), by which time the company or organization handling the security measures should have had more than

enough time to get the set of people on the password database aware of their compromised password.

Alternatively, the mapping of users to their true password is not of much significant use without the associated data of users and their passwords. As long as there is near perfect security on the list of users and their passwords (fake or not), the list of users and their passwords are of no use except identifying system users, which isn't an objective but this still means that our mapping of users to the true password should be kept confidential as used in best practices by many professionals.

(2) Password cracking may become realized to the privacy and security-seeking body as we have an entity attempting many different wrong passwords back-to-back in the case of losing the back-end server (list of users and real, fake password), likely in a short time. It is good practice in this way to have a certain number of incorrect password attempts until the user is made aware of the login attempts, and the company can flag the suspicious activity. In addition, if the attacker inputs a honeyword then the site (network) administrator should be alerted.

With a large enough k-value, and if we can count how many attempts a suspicious body has, then we can find the security-ease of use tradeoff between how many attempts before an entity might start guessing passwords so that close access to the account and after the amount of login attempts the vast (vast) majority of users would need. Note that the amount of resources necessary to brute force may become infeasible if salting or other cryptographic methods were used on the obtained stored data.

(3) We want the honeywords to be statistically distributed as a quite close password to the real one in that it could potentially be the suspected password by an attacker to alert the site administration that a honeyword was guessed on the account. In this sense machine learning models have been offered as a way to create honeywords (scalably and efficiently) which learns human "password-making" characteristics in making honeywords. We can choose different levels of variability and for our case, I would say simply "Bo$tonRedS0x@76!" may provide a good honey word as it makes minute changes but completely changes the general structure of the password. Also, "Bo$tonRedSox1119" as well as "Bo$tonRedSox93S" are very close to the true password, which may actually provide value as it is far enough away that the user likely would not think of it themself, but close enough that a malicious entity might input he honeyword.

(4) I will choose '2001ASpaceOdyssey' as it already appeared in the list of other passwords. Most systems wouldn't allow the password "00…00" and personally I see 1e-5% of the population having a password like "itWb!%s45_3gMoI00286!*mooewTi409##21jUi,", both in length and craziness. For this reason, I thought to choose either of the passwords featuring 'SpaceOdyssey'. I would choose the one with '2001' as it is a year that happened recently and is more relevant to the people in our society. This could be significant for a year of birth, marriage, vacation, or just out of proximity in thought.