Credential Extraction Attacks Against Compromised Credential Checking Services of Password Managers

Yihe Duan, Ding Wang, Yutong Li
College of Cryptology and Cyber Science, Nankai University, Tianjin 300350, China; wangding@nankai.edu.cn
Key Laboratory of Data and Intelligent System Security (NKU), Ministry of Education, Tianjin 300350, China
Tianjin Key Laboratory of Network and Data Security Technology, Nankai University, Tianjin 300350, China

***Abstract*—Password managers (PMs) are highly recommended by security standards and experts to assist users in managing their login credentials. In response to the increasingly serious threat of credential leakages, more and more leading PMs start to leverage third-party compromised credential checking (C3) services, aiming to help users check whether their credentials in the vault have been leaked. C3 services (e.g., Have I Been Pwned) generally maintain extensive datasets of leaked credentials and provide APIs for compromised credential checking. Queries to C3 services comply with k-anonymity security properties, designed to limit information leakage about credentials. However, these queries are deterministic, indicating that identical credentials consistently generate the same query. We find that PMs exhibit identifiable query patterns, such as automatically checking all credentials associated with a single user, and periodically checking users' credentials.**

We, *for the first time*, demonstrate that the query patterns of PMs can be effectively exploited by an honest-but-curious C3 server to identify PM users and extract credentials. We propose a novel credential extraction attack framework based on query pattern leakage to C3 services, and implement attack algorithms targeting PMs' query patterns. Our empirical attacks successfully identify 83.04%∼87.59% of PM users. Furthermore, this query pattern leakage enables attackers to significantly increase the password guessing success rates by 15.42%∼30.43% with one guess, compared to attacks without leveraging this leakage. We evaluate 14 leading PMs, and find that 10 are vulnerable to our attacks. We have disclosed our findings along with recommendations to affected vendors for being aware of (and mitigating) these vulnerabilities.

1. Introduction

Passwords continue to dominate user authentication on the web and show little sign of being replaced in the foreseeable future [2], [4], [14], [15], [22]. The persistent challenges of password management, i.e., the difficulty in creating, remembering, and typing strong passwords, along with growing security concerns, have driven the increasing adoption of password managers [8], [45], [46].

Password managers (PMs) streamline password management by securely storing, generating, and autofilling passwords, and are highly recommended by security standards [1], [3] and experts [34], [57]. The unending password breaches [55] have amplified the threat of credential stuffing [27] and credential tweaking attacks [41], [54], [56]. To mitigate the impact of these threats, PMs leverage compromised credential checking (C3) services to help users detect whether their credentials have been leaked. C3 services, such as the prominent "Have I Been Pwned" (HIBP) [24] and Google Password Checkup [49], generally maintain billions of breached credential records and provide APIs for free checking. For instance, HIBP currently maintains 15 billion records and receives about one billion queries per month [6]. While users can manually query C3 services via websites, PMs streamline the process by automatically sending queries and alerting users to leaked passwords [26].

The query to the C3 server is privacy-preserving, ensuring that the C3 server does not learn the actual content of the queries. For example, Google Password Checkup [49] employs private set intersection protocols to protect users' query information from the C3 server. However, since C3 servers maintain massive breach datasets containing billions of leaked credentials [24], [49], C3 services cannot fully rely on computationally expensive cryptographic protocols. To balance privacy and efficiency, C3 services adopt a k-anonymity bucket mechanism. In this mechanism, the C3 server hashes each credential and groups credentials with the same hash prefix into buckets. During a query, the user sends the hash prefix of a credential to locate the relevant bucket, after which the client uses cryptographic protocols to check whether the credential is leaked within the bucket.

While the bucket mechanism satisfies k-anonymity properties [10], [49], the exposure of hash prefixes still provides an honest-but-curious C3 server with significant advantages in extracting the plaintext of the credentials [32]. Additionally, the bucket mechanism is deterministic, meaning that two queries made with the same credential produce identical results. Consequently, the deterministic nature of the bucket mechanism exposes user query patterns, e.g., frequency, repetition, and periodicity. For instance, if a user periodically queries a credential (u, pw), the server will observe the same l-bit hash prefix $H_l(u, pw)$ each time.

In this work, we, *for the first time*, identify that the query patterns of PMs can directly lead to the exposure

of credentials. This discovery is based on three critical observations, which represent the primary characteristics of PMs' query patterns when interacting with C3 services.

Observation 1. *Password managers query the entirety of users' credentials each time they interact with C3 services.*

PMs provide a function to check all the credentials stored in a PM for a single user (we call it *a credential vault* later). While ordinary users typically query C3 services by manually entering credentials on websites, where the web client sends a single query to the C3 server, PMs behave differently. When users invoke the checking function provided by PMs, the PM repeatedly uses the single-query API to send multiple requests to the C3 server, checking each credential in the vault separately (see Sec. 5.1). Compared to manual input, PMs automatically query all credentials within a short period[1]. As a result, a continuous sequence of queries is likely to originate from a single PM user.

Observation 2. *Password managers repeatedly query users' credentials, which may be triggered by user actions such as adding, deleting, or updating credentials, or by periodic checks performed by the password manager itself.*

In addition to users actively invoking the checking function in PMs, PMs may also perform automatic credential checks without the user's explicit consent. As observed in the query patterns of PMs (see Sec. 5), common user actions such as modifying a single credential can trigger a batch query to the C3 server. As a result, multiple batch queries may be issued by the same PM user within a given period. Moreover, the credential vault typically experiences minimal changes between two consecutive batch queries. Consequently, PMs tend to submit identical or highly similar sets of queries over a given period.

Observation 3. *When password managers query credential vaults for users, credentials within the vault are not deduplicated, resulting in repeated queries for identical credentials.*

Although PMs can help users generate unique passwords for each credential, users rarely rely exclusively on randomly generated passwords [35]. Instead, they tend to create passwords manually [12], [39], [60]. If users exhibit password reuse habits, the vault will contain the same passwords, and PMs will repeatedly query the same passwords in a batch query. Though queries only consist of hash prefixes, a short consecutive sequence of queries is unlikely to have same hash prefixes from different credentials. For example, for a 20-bit hash prefix of SHA256 and a query sequence of length 100, the probability of a collision between any two queries is approximately 0.47%. As a result, repeatedly queried hash prefixes in a short sequence of queries are most likely associated with a single PM user[2].

We propose a novel attack exploiting PM query pattern to extract the credentials of queries issued by PM users. The credential extraction attacks comprise two stages: identification and extraction. In the identification stage, the attacker determines which queries originate from the same user. This enables each query to be linked with others in its context. In the extraction stage, the attacker leverages the contextual information to locate plaintext credentials within a breach dataset. If the plaintext of a query is not directly found, the attacker can guess it using the plaintexts of other queries within the same context. Building on the above three key observations about the characteristics of PMs' query patterns, we implement attack algorithms to exploit PM users' credentials. To demonstrate the effectiveness of our attack, we conduct comprehensive empirical evaluations. Additionally, to highlight its practicality, we manually analyze the query patterns of 14 popular PMs.

Li et al. [32] demonstrated that leveraging users' leaked passwords from breach datasets can significantly improve the success rate of credential extraction attacks. However, their approach assumes that attackers can obtain users' identities through methods such as XSS scripting or phishing to locate the users' leaked passwords in the breach dataset. In contrast, our work shows that by *merely* leveraging query pattern leakage, credential extraction attack can identify users without interacting with users through additional channels like XSS scripts or phishing emails. Our findings highlight that the leakage of just a few bits of hash prefixes can have a much greater impact than previously expected/believed. In summary, our contributions are as follows:

- We propose a novel credential extraction attack framework leveraging the query pattern leakage in C3 protocols. Our attack consists of two phases, i.e., identification and extraction, with four modules: l-identifying, range combining, credential connecting, and credential guessing. We provide instances on attacking PMs' query patterns to demonstrate the effectiveness of our attack. We also formalize the interfaces of the four modules to generalize the framework for diverse query patterns.
- We empirically evaluate our proposed attack by simulating queries on two real-world datasets [5], [16], comprising approximately 1.5 billion leaked credentials. The results show that the identification phase can successfully identify PM users' queries with accuracy rates of 87.59% for the username-based mechanism, 83.89% for the password-based mechanism, and 83.04% for the credential-based bucket mechanism. The extraction phase achieves password guessing success rates ranging from 47.14% to 95.21% within 10^3 guesses. Notably, with only one guess, it outperforms attacks that do not leverage query pattern leakage by 17.34% to 30.43%.
- To gain deeper insights into our attack, we design three additional scenarios to evaluate the impact of leaked password updates, network instability, and dynamic credential vaults. Our results demonstrate that timely updating of leaked passwords offers a degree of protection against query pattern leakage attacks. In addition, while unstable network conditions can impair the accuracy of identification, their

1. We refer to the set of queries triggered during a single invocation of the checking function by a password manager as *a batch query*.

2. Different users may use the same passwords, so we leverage the observation 3 carefully (see more discussion in Sec. 3.1).

impact on the extraction is limited.

- We investigate the query patterns of 14 popular PMs in the wild. Our findings reveal that 13 out of 14 PMs query the entire vault with each batch query, confirming the practicality of our attack. Additionally, we show that the trigger conditions for queries are closely tied to user actions, such as adding, changing, or deleting credentials. This means that identified batch queries could be used to infer and monitor specific user actions. Alarmingly, 8 out of 14 PMs embed cookies in their queries, which significantly exacerbates identification. We have disclosed these vulnerabilities to the respective vendors, provided recommendations for mitigation, and received positive response (see Sec. 5).

2. Preliminaries

2.1. The security of password managers

The security of PMs has received significant attention. With the increasing prevalence of cloud-based PMs, studies [13], [17], [19], [21], [38], [58], [59] have focused on end-to-end encryption schemes to ensure the security of encrypted vaults, yielding important findings: unencrypted metadata can enable distinguishing and tampering attacks [19], [21], [38], and relying solely on a master password cannot mitigate the risk of offline guessing attacks [13], [17], [19], [58], [59]. These insights have informed secure approaches for encrypting and storing credential vaults.

In end-to-end encryption schemes, PMs perform core functionalities, such as autofill and password generation, on plaintext data after client-side authentication and decryption. Since these operations occur locally and do not interact with the server, they may appear free of additional risks. However, automation in password management can unintentionally introduce new vulnerabilities. Silver et al. [47] and Stock and Johns [48] discovered that autofill can leak passwords if websites are vulnerable to network injection or XSS attacks. Oesch and Ruoti [38] demonstrated at USENIX SEC'20 that autofilling into iframes is susceptible to clickjacking attacks. Later, Oesch et al. [37] evaluated mobile autofill frameworks at ACSAC'21 and found that mobile PMs are less secure than their desktop counterparts. Lin et al. [33] and Fu and Wang [20] revealed that autofill can input passwords into hidden forms, potentially leaking passwords without user awareness.

Existing research has primarily focused on autofill functions. To the best of our knowledge, *we are the first to reveal vulnerabilities in automatic password-checking mechanisms*.

2.2. Overview of C3 services

Compromised Credentials Checking (C3) services are web-based services that help users check whether their account credentials have been compromised. These services aggregate leaked credentials from public breaches [24] or private data sharing by government organizations [25]. C3 services support various query types, allowing users to check different aspects of their credentials. Specifically, users can query their credentials to determine if credentials have been leaked, or query specific usernames or passwords to check if those have been compromised. Additionally, some C3 services, such as "Have I Been Pwned" (HIBP) [24], enable users to check website breaches by querying domains. In 2022, Pal et al. [42] introduced a second generation of C3 services, called "Might I Get Pwned" (MIGP), which detects whether users' credentials are leaked or semantically similar to breached credentials. This service is now integrated into Cloudflare's Web Application Firewall (WAF) [50].

C3 services are available on websites, such as HIBP, and have also been integrated into PMs. For example, 1Password utilizes HIBP's API to check whether passwords stored in vaults are compromised [7], while LastPass uses Enzoic for monitoring [29]. Additionally, browser-based PMs, such as those in Chrome [49] and Edge [30], support their own C3 services. For clarity, we define users who manually query credentials on websites as *Ordinary Users*, and those who rely on PMs for automatic credential checks as *PM Users*.

2.3. Overview of C3 protocols

C3 services are the platforms providing credential checking functions, while C3 protocols define the interaction between clients and C3 servers to enable these functionalities.

Basic settings. In the C3 protocol, the server maintains a *breach dataset* comprising a set S of M username-password pairs $(u_1, pw_1), \ldots, (u_M, pw_M)$ where M can be more than ten billions [24]. Users can query only the username $(u^i, *) \in S$, only the password $(*, pw^i) \in S$, or both username and password $(u^i, pw^i) \in S$.

The credential vault is a dynamic set of a user's credentials, denoted as $(u^1, pw^1), \ldots, (u^m, pw^m)$, with a size m at time t. Both ordinary users and PM users maintain such credential vaults, but PM users manage all credentials within PMs and use C3 services via PMs.

Bucket mechanisms. Given the massive number of leaked credentials in breach dataset S, C3 protocols employ a bucket mechanism to partition the S into multiple buckets. Using this mechanism, the client first sends an identifier to locate the corresponding bucket. The server and client then execute privacy-preserving protocols within that bucket. The identifier is typically a hash prefix of the username and/or password, ensuring that credentials with the same hash prefix are grouped into the same bucket. Bucket mechanisms allow C3 services to perform privacy-preserving compromised-credential checkups within a plausible data size, but they also leak a small amount of information about users' queries. For example, if a C3 service uses a password-based bucket mechanism, a small portion of the password hash (e.g., 20 bits) will be leaked to the server.

Hash comparison. "Have I Been Pwned" (HIBP) [24] employs a simple protocol. The server preprocesses the breach dataset S by computing the SHA-1 hash for each

TABLE 1: Attacks on C3 protocols with different leakage.

Attack type	Attack originator	Attack goal	C3 Leakage leveraged	Existing literature
Breach Extraction	Malicious C3 Client	Private breach information[1]	τ-similarity	Pal et al. [42]
			τ-similarity& τ-collision[2]	Pasquini et al. [43]
Credential Extraction	Honest-but-curious C3 Server	Query plaintext(i.e., user credential)	Hash prefix	Li et al. [32][3]
			Hash prefix& Query pattern	This work

[1] The C3 server may obtain private breach datasets, such as those shared by government agencies [25], which should remain unknown to clients.
[2] τ is a function that generates similar passwords from a given input password. In MIGP [42], τ may produce the same output for different inputs.
[3] In [32], the attacker is assumed to obtain the usernames corresponding to the queries through techniques such as XSS scripting or phishing.

password. When the client sends a hash prefix to locate a bucket, the server returns all hashed passwords within that bucket. The client then computes the SHA-1 hash of its password and checks for a match in the bucket. This scheme does not protect the data stored on the server, allowing the client to guess passwords within a bucket.

Two-party secure computation protocols. To protect the breach dataset stored on the server, the client and server can employ two-party secure computation protocols. Google Password Checkup [49] adopts an OPRF-based private set intersection (PSI) protocol. Briefly, the server preprocesses the breach dataset by hashing each credential $H(u_i, pw_i)$ and protecting the result with a private key k, $H(u_i, pw_i)^k$. During interaction, the client blinds the query by selecting a random secret ρ and sends $X = H(u, pw)^\rho$ to the server. The server then computes $Y = X^k$ and returns it along with all preprocessed credentials in the bucket. The client calculates $x' = Y^{\frac{1}{\rho}}$ and checks whether x' exists in the bucket. By using a formally proven secure PSI protocol, this C3 protocol ensures privacy for both the client and the server within the bucket. Furthermore, with the bucket mechanism, the C3 protocol can incorporate other secure computation techniques, such as homomorphic encryption [30], [31].

Attacks on C3 protocols. As summarized in Table 1, existing attacks on C3 protocols fall into two categories. Breach extraction attacks are launched by malicious clients attempting to extract private breach data from the server. In MIGP [42], the server computes a similarity function τ that determines whether a client's query is semantically similar to passwords in the breach dataset. A malicious client can exploit the similarity feedback to gradually extract breach contents. At IEEE S&P 2024, Pasquini et al. [43] discovered that τ may map multiple distinct inputs to the same output password, creating collisions that can be further exploited to extract breach entries more efficiently.

In our work, we focus on credential extraction attacks launched by an honest-but-curious server, which aims to extract the plaintext credentials behind user queries. Li et al. [32] demonstrated that the hash prefix leakage in password-based bucket mechanisms could increase password guessing success rates by up to 12x. However, their attack assumes that the server already knows the usernames behind the queries, obtained through techniques such as XSS or phishing. In contrast, our work shows that query pattern leakage alone can reveal the users' identities, thereby making credential extraction attacks significantly more practical and realistic in real-world C3 deployments.

2.4. Threat model

Attacker's capabilities. The attacker in the credential extraction attack is modeled as an honest-but-curious C3 server that receives client queries and arranges them chronologically based on their transmission time. During the interaction, the attacker gains information leaked by the bucket mechanism but does not learn the results of the credential checks. Since the communication channel is protected by TLS, network attackers are not a concern. We assume that the server has access to the plaintext breach dataset, as all C3 protocols require preprocessing on plaintext credentials, despite using only ciphertexts during runtime. Our threat model follows that of Li et al. [32], except that we do not assume the attacker can obtain the usernames behind the queries through techniques such as XSS or phishing. In our setting, all information available to the attacker comes solely from the queries and the breach dataset.

Attacker's goals. The attacker aims to extract the plaintext corresponding to a C3 query. The attacker generates plaintext guesses according to the bucketization scheme (i.e., usernames and/or passwords under username-, password-, or credential-based bucketization). The attacker outputs guesses of plaintext, then verifies the guesses online. Given the limitations imposed by online guessing defenses, such as rate limiting [9], [11], the attacker aims to identify the correct plaintext using as few guesses as possible [41], [53], [56]. In this work, we focus on credential extraction attacks that leverage the query patterns of PM users. Attacks targeting ordinary users are beyond the scope of this study and are left for future work.

3. Credential Extraction Attacks

In this section, we present a novel credential extraction attack framework that exploits users' query pattern leakage to extract their credentials. Our attack framework is generic and can be applied to extract credentials by leveraging query patterns. In this work, we demonstrate its application to PM users. The honest-but-curious C3 server first collects a series of user queries. Each day, the server processes billions of queries [49] and organizes them by their receiving timestamps. After collecting a series of user queries, the attacker can perform credential extraction attack. Fig. 1 shows the overview of the attack with an example.

The credential extraction attacks on PM users contain two phases: identification and extraction. The identification phase aims to cluster queries and determine which ones originate from the same PM users. The result of the identification phase is a set of *identified subsequences*, comprising multiple clusters of hash prefixes attributed to individual PM users. After identification, the attacker builds the relations between queries with their context. Extraction phase aims to

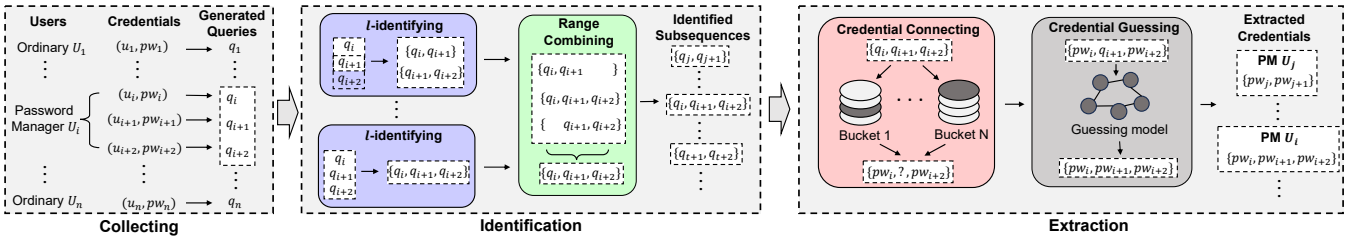


Figure 1: Overview of our credential extraction attack with an example. The attacker, modeled as an honest-but-curious C3 server, collects a sequence of queries $q_1, \ldots, q_n$, where q_i, q_{i+1}, and q_{i+2} may originate from the same PM user. In the identification phase, the attacker applies l-identifying (e.g., with $l = 2, 3$) to identify and merge subsequences belonging to the same user. In the extraction phase, if the buckets of q_i and q_{i+2} contain credentials from the same breached user, their plaintexts pw_i and pw_{i+2} are connected. For unconnected queries q_{i+1}, the attacker uses connected credentials to guess it.

reconstruct the plaintext of queries. It leverages the context information built by the identification phase and breach datasets to recover the plaintext. Specifically, the breach datasets contain information about which leaked credentials are from the same user. Therefore, the relation between queries with their context are probably connected to the credentials from one user in breach dataset, and then help extract the plaintext of queries. After extraction, for each query in the identified subsequences, the attack outputs the guesses of the corresponding plaintext of the hash prefix.

3.1. Identification phase

As depicted in Fig. 1, the identification phase consists of two modules: *l-identifying* and *range combining*. The l-identifying algorithm identifies all query sequences of length l in which each query is potentially issued by the same PM user. Since l-identifying only handles query sequences of a specific length, multiple runs with different l values are required. The results from different l-identifying runs are then combined, as larger l values may subsume smaller ones. For example, 3-identifying may recognize query sequence q_i, q_{i+1}, q_{i+2}, while 2-identifying recognizes q_i, q_{i+1}. In this case, the result of 2-identifying is merged into 3-identifying.

l-identifying. We define the l-identifying algorithm as a function $lIdentifying(Seq, l)$, which takes two inputs: a query series Seq and a length parameter l, and outputs all subsequences of length l that satisfy specific query pattern, and we call these subsequences *identified subsequences*. As stated in observation 2 in Sec. 1, PMs periodically query users' passwords, their repetitive nature allows the l-identifying algorithm to be treated as a repeated-subsequence identification problem.

In Algo. 1, we provide our instantiation of $lIdentifying$ algorithm. We maintain a table to record the occurrence frequency of each subsequence. To distinguish PM users' queries, we set a threshold min_count to filter out less frequent subsequences. According to Theorem 1 in Appendix A, queries from ordinary users are unlikely to form frequently occurring subsequences, which means that queries from PM users are high likely to remain.

Additionally, based on observation 3 in Sec. 1, subsequences containing repeated queries are assigned higher weights w to help them surpass the identification threshold, as they are more likely to originate from PM users. However, since different users may use identical passwords, particularly popular ones, this repetition could increase the false positive rate. Therefore, we choose a relatively small weight, setting $w = 0.5$ in our evaluation for simplicity and clarity. To further refine its usage, w can be adjusted based on the likelihood that a query originates from a popular password. For example, we can assign $w = 0$ to repeated queries with hash prefixes corresponding to popular passwords, while retaining $w = 0.5$ for other cases.

Algorithm 1: l-identifying attack $lIdentifying$.

Input: A query series Seq consisting of N queries, and a length l.
Output: A set T of subsequences, each of length l.

```
T ← EmptySet(); //Initialize T to an empty set.
original_subseq ← ExtractAllSubseq(Seq, l);
//Extract N − l + 1 subsequences.
subseq_table ← GenerateTablewithCount();
//Generate a hash table with default values initialized to 0.
for i ← 1 to N − l + 1 do
    subseq_table[sorted(orginal_subseq[i])] += 1;
    if orginal_subseq has repeated queries then
        subseq_table[sorted(orginal_subseq[i])] += w;
        //If a subsequence contains repeated queries, it is given
        additional weight for identification.
foreach item in subseq_table do
    if item.value >= min_count then
        T.add(item.key);
return T
```

Queries are sorted before recording because we want sequences with the same queries to be treated as equivalent, so the specific order does not matter. And the sorting process offers two benefits: (1) it mitigates issues caused by out-of-order query arrivals; and (2) it bypasses heuristic defenses, such as randomizing query orders.

Given that l is much smaller than the length of the query sequence Seq, the time complexity of the algorithm is linear, $O(N)$, making it highly efficient. The space complexity is also $O(N)$. To reduce space usage, the attacker can adjust the min_count parameter to prune results.

Range combining. After l-identifying, the attacker obtains subsequences of a single length. To derive subsequences of varying lengths, the attacker runs l-identifying with different l parameters. We define the range combining as

a function $RCombining(\hat{T}, low, high)$, which takes a set $\hat{T}$ of subsequence with spanning lengths from low to $high$, i.e., $T_{low} \dots T_{high}$. The $RCombining$ function aims to prune the small length subsequences which are covered into longer subsequences. After $RCombining$, the attacker gets a set T' with different length of identified subsequences.

In Algo. 2, we first initialize a set T' and iterate over subsequences from shortest to longest. A shorter subsequence is initially added to T', but it may be removed if its queries are fully contained within a longer one.

The time complexity of the range combining algorithm is linear with respect to the total length of all collected queries. Since the algorithm iterates over each identified query subset, its time complexity is $O(N \cdot (high - low))$. Notably, the range combining algorithm can be integrated with the l-identifying algorithm. By running the l-identifying algorithm sequentially from low l to high l, we can add results to T' and combine them on the fly, reducing overall time costs. Additional space is required to store T', resulting in an extra space complexity of $O(N \cdot (high - low))$.

Algorithm 2: Range combining $RCombining$.

Input: A set $\hat{T}$ contains subsequences $T_{low} \dots T_{high}$ of varying lengths, ranging from l_{low} to l_{high}.
Output: A set T' of subsequences with lengths $l_{low} \sim l_{high}$.

```
T' ← GenerateSet(); T' ← T_low;
foreach T_i from T_{low+1} to T_high do
    l_i ← T_i.length();
    l_{i-1} ← T_{i-1}.length();
    foreach subseq in T_i do
        T'.add(subseq); //Add the subsequence to the set.
        lower_seq ← ExtractSubsequences(subseq, l_{i-1});
        //Extract all subsets of subseq with a length of l_{i-1}.
        foreach s in lower_seq do
            if s in T' then
                T'.remove(s);
return T'
```

3.2. Extraction phase

After the identification phase, the attacker obtains a set T' of identified subsequences, each of which is assumed to originate from a single user. The attacker then proceeds to extract the credentials corresponding to these queries. As shown in Fig. 1, extraction phase consists of two modules: credential connecting and credential guessing. The attacker first utilizes the breached dataset to connect queries with plaintext credentials. For example, if two queries targeting different buckets match two credentials belonging to the same user in the breach dataset, the attacker can connect these queries to corresponding plaintexts. For queries that cannot be directly connected, the attacker then uses the connected plaintext credentials to guess the unknown queries.

Credential connecting. The attacker possesses the breach dataset S, containing the plaintext of credentials. After obtaining queries, the attacker can use user queries to locate the corresponding buckets and retrieve the plaintext from the buckets. We define the credential connecting as a function $CreConnect(T', S)$, where S is a set of leaked credentials $(u_1, pw_1), \dots, (u_M, pw_M)$, and T' is a set of identified subsequences from different PM users. The $CreConnect$ function outputs a new set T'' of identified subsequences, where partial queries are resolved into plaintext form.

Algo. 3 leverages an inverted index table to efficiently identify the best credential matches. The attacker first joins credentials belonging to the same user in the breach dataset S. In our instantiation, we apply the mixed method [41] to join credentials. This method assumes that credentials sharing the same username prefix, i.e., email address, and at least one common password, belong to the same user. Next, the attacker calculates the hash prefix of each credential.

We set a threshold $min\_overlap$ to define the minimum matches required between hash prefixes in the breach dataset and identified subsequences. The $min\_overlap$ is set to at least two, as only identified subsequences that connect at least two different buckets with credentials may provide useful information. The attacker begins by generating a lookup table cre_index for the hash prefixes in the breach dataset $hash\_S$, storing the location of each unique hash prefix. For each identified subsequence, the attacker checks which credentials in the breach dataset share the same hash prefix. Next, the attacker selects the users in the breach dataset with the highest count of connecting hash prefixes to the identified subsequences. Finally, the attacker uses the plaintext corresponding to the connecting hash prefixes and assigns these plaintexts as the correct results for the queries.

Algorithm 3: Credential connecting $CreConnect$.

Input: A set T' of identified subsequences from different PM users. For a specific subsequence ss_i, T' includes a list of queries $q_{i_1}, \dots, q_{i_m}$. A breach dataset S consists of credentials $\langle u_1, pw_1 \rangle, \dots, \langle u_M, pw_M \rangle$.
Output: A set T'' of identified subsequences from different PM users, where the queries of a specific subsequence ss_i are partially recovered as $p_1, \dots, p_n$, with $n \leq m$.

```
S' ← UserJoin(S); //Join credentials belonging to the same user.
hash_S ← BucketizationAll(S');//Calculate the hash prefix for
  each credential in S' to connect with identified queries.
cre_index ← GenerateTablewithList();
//Create a inverted index table to record credentials in S'.
foreach C_i in hash_S do
    foreach hash_cre_j in C_i do
        cre_index[hash_cre_j].append(i);
foreach ss_i in T' do
    overlap_count ← GenerateTablewithCount();
    //Create a table to record the overlap between T' and S'.
    foreach q_j in ss_i do
        if q_j in cre_index then
            foreach idx in cre_index[q_j] do
                overlap_count[idx] += 1;
    Find the maximum value max_overlap in overlap_count;
    if max_overlap ≥ min_overlap then
        Locate the index of max_overlap in overlap_count to
          obtain the best_match in hash_S;
        //If multiple maximum values exist, select one of them.
        Use S' to retrieve the plaintext of best_match;
        Replace the overlapping part of ss_i with the plaintext of
          best_match, and update ss_i;
All identified queries are updated in T', resulting in a new set T'';
return T''
```

The $UserJoin$ and $BucketizationAll$ functions can be pre-processed offline. During the online attack phase, the use

of an inverted index table yields a time complexity of $O(N+M)$ and a space complexity of $O(M)$, where N denotes the number of identified subsequences and M represents the number of credentials in the breach dataset.

Credential guessing. After credential connecting, the attacker connects identified subsequences to plaintexts in the breach dataset, partially extracting queries and retrieving other leaked credentials in the breach dataset from those users. However, some hash prefixes may still remain unrecovered. To address this, the attacker uses the known plaintext and hash prefixes to guess the unknown queries. We define the credential guessing as a function $CreGuessing(T'', S)$, where T'' is a set of identified subsequences with partial plaintext, and S is the breach dataset used to train the guessing model. The function outputs the guesses of all identified subsequences.

In Algo. 4, the attacker processes a set of identified subsequences. For each subsequence, the attacker uses the connected plaintext to generate guesses and the hash prefix to eliminate incorrect guesses. For example, under username-based bucketization, the attacker can use connected usernames to guess a user's other unconnected usernames. Credential connecting applies only to credentials already present in the breach dataset, whereas credential guessing can target unleaked credentials that are not in the dataset. It should be noted that the effectiveness of guessing unleaked credentials depends on the underlying guessing model. For instance, under password-based bucketization, current targeted password guessing models are proficient at guessing popular, reused, and passwords similar to previously used ones [41], [54], [56]. Consequently, our credential guessing algorithm under password-based bucketization is effective primarily for passwords with such characteristics.

Algorithm 4: Credentials guessing $CreGuessing$.

Input: A set T'' of identified subsequences from users. For a specific subsequence ss_i, T'' includes the connected plaintext $p_{i_1}, \ldots, p_{i_n}$ and a list of unresolved queries $q_{i_1}, \ldots, q_{i_t}$. A breach dataset S contains credentials $(u_1, pw_1), \ldots, (u_M, pw_M)$.
Output: A set P of identified subsequences, where for each subsequence ss_i, P contains guesses $p_{i_1}, \ldots, p_{i_m}$.

```
rGM ← GuessingModelInit(S);
//Train a guessing model using the S for different bucketizations (e.g., guessing usernames under username-based bucketization).
foreach U_i in T'' do
    old_cre ← [p_{i_1} ... p_{i_n}] + other matched credentials;
    unknown_prefix ← [q_{i_1} ... q_{i_t}];
    guess_list ← InitializeList(); //Initialize a list to combine all guesses for each plaintext.
    while guess_list.length() < guess_num do
        foreach cre in old_cre do
            guesses ← rGM.gen(cre.pwd, num_q);
            //Generate num_q new guesses from one plaintext, ranked from high to low probability.
            foreach guess in guesses do
                hguess ← Bucktization(guess);
                if hguess not in unknown_prefix then
                    guesses.remove(guess);
            guess_list.append(guesses);
    Update U_i in T'' with [p_{i_1} ... p_{i_n}] + guess_list;
All recovered queries are updated in T'', resulting in a new set P;
return P
```

We define two parameters, $guess\_num$ and num_q, to control the guessing efficiency. The parameter num_q specifies the number of guesses generated in each round, while $guess\_num$ determines when to terminate the generation of guess lists. The complexity of $CreGuessing$ depends on the guessing model's complexity and the values of $guess\_num$ and num_q. Larger values of $guess\_num$ and num_q lead to increased time and space costs.

4. Evaluation

In this section, we present an empirical evaluation of our attacks. Since no real datasets of C3 service queries are available, we first propose a methodology to simulate queries with different scenarios. We use two real-world breach datasets: 4iQ [16] and Naz.API [5]. 4iQ is a breach dataset containing 1.4 billion credentials which has been used in previous research [32], [41], [42] and Naz.API contains 300 million credentials and it contains saved logins and passwords in browser PMs which is more suitable for our setting. We partition breach datasets into two parts: 90% simulates the C3 service's leaked dataset, while the remaining 10% forms the test dataset. All credentials in the C3 service's leaked dataset are considered *leaked credentials*. If a credential in the test dataset appears in the C3 leaked dataset, it is classified as *leaked*; otherwise, it is classified as *unleaked*. From the test dataset, users with multiple passwords are classified as PM users, while others are considered ordinary users. Details on the dataset information are in Appendix B.

Ethical considerations. Although these datasets are publicly accessible, we take extra precautions to protect private data. All processing is conducted on computers that are not connected to the Internet, only aggregated statistics are reported, and each credential is treated as confidential.

Open science. To ensure the reproducibility of our work, we have open-sourced our artifact to the community at https://tinyurl.com/6ffptncc . Our artifact includes the code for query simulation described in Section 4.1, as well as attack examples. We do not open the datasets or the password guessing model, as they are not part of our contributions and are publicly accessible [16], [56].

4.1. Evaluation setup

We randomly sample credentials from the test dataset to construct the query series Q. We implement three bucket mechanisms, i.e., username-based, password-based, and credential-based bucket mechanisms respectively. For ordinary users, the hash prefix of the selected username and/or password is computed and added to Q. For PM users, hash prefixes of all usernames and/or passwords in their vaults are computed and appended to Q.

To gain deeper insights, we construct four scenarios to simulate different real-world situations (see Table 2 for details). These scenarios vary based on four key parameters:

- %Asyn: The %Asyn parameter indicates whether the query order from the client matches the order

TABLE 2: Simulation of different scenarios.

Scenario	%Asyn	%Clean	%Intercept	%Active
# 1	0%	0%	0%	0%
# 2	100%	100%	10%	1%
# 3	100%	0%	10%	1%
# 4	100%	0%	50%	1%

received by the C3 server. It simulates real-world asynchrony by randomly shuffling and sending all credentials of one PM user.

- %Clean: The %Clean parameter indicates whether PM users change their passwords after detecting a leak. We make no assumptions about weak human-preference actions, such as reusing an uncompromised password or creating a similar one. Instead, we assume users generate new random passwords.
- %Intercept: The %Intercept parameter indicates whether ordinary users' queries are interspersed among PM users' queries. This parameter simulates a concurrency network environment. In the simulation, for each query from a PM user, we set a probability %Intercept that an ordinary user sends a query immediately afterward.
- %Active: The %Active parameter indicates whether PM users actively manage their vaults by adding, deleting, or modifying credentials. Higher values of this parameter correspond to more frequent changes in vaults. In our simulation, for each batch query of PMs, we set a probability %Active that a PM user updates their vault after query. These updates simulate routine credential management behavior and are independent of leaked passwords.

The basic setting is Scenario #1. In this scenario, all credentials in the vault are sent to the C3 server in order each time, and the server receives the queries in the same order. Additionally, the vaults of PM users remain static during each batch query. This scenario represents a relatively simple situation. However, it is ideal for illustrating the core principles of the attack, as the static and ordered nature of the queries maximizes the information leakage exploited by our attacks. Additionally, it is realistic to some extent, as users may query their credential vaults multiple times without making changes. For instance, KeePass queries the entire credential vault on each login, even if the vault remains unchanged (see Table 5 for details). Consequently, if users ignore alerts, their vaults will continue to be sent to the C3 service in this static mode.

In this Scenario #2, we simulate a concurrent environment where ordinary users' queries may intersperse with a set of PM queries sent to the C3 server. We set %Intercept = 10% to model this interaction, and randomly shuffle all credentials of PM users. Additionally, for each batch query of PMs, PM users have a %Active = 1% probability of updating their vaults. When detecting a leak, PM users update their credentials with a %Clean = 100%.

Scenario #3 modifies Scenario #2 by assuming that PM users do not update their leaked credentials i.e., %Clean = 0%. This adjustment isolates the impact of changing leaked passwords, providing insights into the role of encouraging users to update leaked credentials. Scenario #4 modifies Scenario #3 by increasing %Intercept to 50%, simulating a more unstable environment.

Evaluation metrics. We evaluate the credential extraction attack the same as other research on targeted online guessing attacks [32], [41], [53], [56], i.e., the success guess rate within a limited number of guesses. To further understand the credential extraction attack, we define several metrics to conduct ablation experiments on the identification and credential connecting in the Sec. 4.3 and 4.4.

Comparison. To evaluate the effectiveness of our credential extraction attack, we compare it with a basic attack and the attack proposed by Li et al. [32]. In the basic attack, the attacker relies solely on the password distribution from the breach dataset for guessing, and uses hash prefixes to eliminate incorrect guesses. The attack by Li et al. [32] assumes that the attacker can obtain the username associated with a query through methods such as XSS scripting or phishing emails. With the username, the attacker can retrieve all leaked credentials associated with the user corresponding to the query. In contrast, our attack incurs lower cost and requires fewer assumptions than that of Li et al [32]. Since their approach represents the upper bound of guessing success rates, we use it as a benchmark to assess the gap.

Experimental environment. We conduct our experiments on a workstation equipped with an Intel Xeon Gold processor and an NVIDIA GeForce RTX 3090 GPU (24 GB VRAM). We focus on PM users with 10-20 credentials (representing 26,428 users with 361,927 credentials in the 4iQ dataset and 61,423 users with 782,058 credentials in the Naz.API dataset). Our experimental datasets are not composed exclusively of PM users, and the distribution of credentials per user is highly skewed: 98% of users in 4iQ and 86% in Naz.API hold only 2-5 credentials. Therefore, selecting users with fewer credentials, e.g., 2-5 credentials, likely misclassifies many real-world non-PM users as PM users in our simulation experiments. Moreover, analyzing users with very few credentials would generate numerous subsequences, increasing computational overhead without yielding additional insights or benefits. At the other end of the spectrum, users with more than 20 credentials are too rare to provide a meaningful sample size.

4.2. Evaluation results

We perform l-identifying on query sequences of length 10 to 20, and then apply range combining to obtain a set of identified subsequences. These subsequences are matched with users in the breach dataset based on at least two overlapping hash prefixes, after which we attempt to guess the remaining unconnected credentials. In this section, we focus on attacks against the password-based bucketization, while results for the username-based and credential-based bucketizations are provided in Appendix C.

For the basic attack, password guesses are generated using a 3-gram Markov model trained on the passwords from

TABLE 3: Guessing success rates under password-based bucketization for PM users with 10-20 credentials.[†]

Scenario	Attack	Leaked				Unleaked				Total			
		$q=1$	$q=10$	$q=10^2$	$q=10^3$	$q=1$	$q=10$	$q=10^2$	$q=10^3$	$q=1$	$q=10$	$q=10^2$	$q=10^3$
#1: 4iQ	Basic	80.49%	87.17%	90.36%	92.48%	0.91%	4.94%	12.00%	20.12%	73.43%	79.88%	83.41%	86.06%
#1: 4iQ	Li et al. [32]	87.68%	90.67%	92.85%	94.44%	17.35%	20.14%	27.11%	34.09%	81.71%	84.68%	87.27%	89.31%
#1: 4iQ	Ours	92.59%	94.36%	95.46%	96.67%	3.31%	10.33%	19.42%	25.27%	**90.77%**	**92.64%**	**93.92%**	**95.21%**
#2: 4iQ	Basic	-	-	-	-	0.00%	0.01%	0.17%	0.44%	0.00%	0.01%	0.17%	0.44%
#2: 4iQ	Li et al. [32]	-	-	-	-	1.03%	1.03%	1.10%	1.41%	1.03%	1.03%	1.10%	1.41%
#2: 4iQ	Ours	-	-	-	-	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%
#3: 4iQ	Basic	79.96%	86.54%	89.84%	92.08%	0.79%	4.43%	11.34%	18.60%	72.71%	79.01%	82.63%	85.35%
#3: 4iQ	Li et al. [32]	86.98%	90.05%	92.62%	94.14%	18.12%	21.37%	26.07%	31.20%	81.03%	84.12%	86.87%	88.70%
#3: 4iQ	Ours	89.60%	92.18%	93.61%	95.18%	1.04%	4.66%	8.81%	15.03%	88.12%	**90.73%**	**92.19%**	**93.81%**
#4: 4iQ	Basic	79.93%	86.76%	90.07%	92.18%	0.81%	4.71%	11.78%	19.82%	72.65%	79.21%	82.86%	85.52%
#4: 4iQ	Li et al. [32]	88.10%	90.67%	93.00%	94.26%	12.87%	15.70%	20.51%	26.58%	81.49%	84.09%	86.64%	88.31%
#4: 4iQ	Ours	91.91%	94.01%	95.33%	96.28%	0.68%	5.78%	14.63%	22.79%	89.66%	**91.84%**	**93.34%**	**94.46%**
#1: Naz	Basic	4.43%	10.01%	17.64%	26.20%	0.55%	2.28%	5.01%	8.40%	4.01%	9.28%	16.46%	24.54%
#1: Naz	Li et al. [32]	89.15%	89.63%	90.88%	91.79%	14.27%	16.17%	18.16%	20.06%	83.23%	83.82%	85.13%	86.12%
#1: Naz	Ours	37.66%	41.09%	45.74%	50.86%	3.47%	4.93%	7.66%	11.31%	34.44%	37.68%	42.16%	47.14%
#2: Naz	Basic	-	-	-	-	0.01%	0.05%	0.18%	0.36%	0.01%	0.05%	0.18%	0.36%
#2: Naz	Li et al. [32]	-	-	-	-	1.01%	1.05%	1.15%	1.28%	1.01%	1.05%	1.15%	1.28%
#2: Naz	Ours	-	-	-	-	0.00%	0.07%	0.21%	0.35%	0.00%	0.07%	0.21%	0.35%
#3: Naz	Basic	4.40%	10.09%	17.49%	26.17%	0.42%	1.91%	4.17%	6.90%	3.95%	9.16%	15.98%	23.99%
#3: Naz	Li et al. [32]	87.91%	88.85%	89.42%	90.04%	12.17%	13.12%	14.60%	17.16%	79.15%	80.09%	80.77%	81.62%
#3: Naz	Ours	24.51%	29.28%	35.25%	43.25%	3.31%	4.74%	7.15%	9.64%	22.26%	26.68%	32.27%	39.68%
#4: Naz	Basic	4.29%	9.98%	17.44%	26.07%	0.43%	1.76%	3.89%	6.57%	3.85%	9.03%	15.87%	23.82%
#4: Naz	Li et al. [32]	89.10%	89.71%	90.36%	91.35%	11.62%	12.62%	14.12%	16.48%	80.56%	81.22%	81.96%	83.10%
#4: Naz	Ours	34.39%	38.56%	43.85%	50.43%	4.07%	5.45%	6.55%	8.80%	31.02%	34.88%	39.70%	45.80%

[†] "-" indicates that the denominator of the rate is zero; Bold values highlight results that exceed 90%. q is the guess number. The results show that our attack on the password-based bucketization is particularly effective at extracting credentials from leaked passwords of lower popularity.

the breach dataset. For our attack and the one proposed by Li et al. [32], the guesses are composed of three components: (1) leaked credentials of connected users from the breach dataset, (2) guesses generated using the PointerGuess password-guessing model [56] based on users' leaked credentials, and (3) guesses generated using the 3-gram Markov model trained on the breach dataset. For guesses generated by PointerGuess, we use each leaked password of connected users to produce 2^{11} candidates, and adopt the heuristic method from [41] to combine results from single-old-password guesses. To the best of our knowledge, all targeted password-guessing models cannot support multiple old passwords as input. Even MS-PointerGuess [56], one of the most advanced models, is limited to handling at most two old passwords, while our scenario may involve more old passwords as input. Additionally, we generate 700 million guesses using the Markov model. Hash prefixes are used to eliminate incorrect guesses. We evaluate the final guessing success rates at different thresholds: $guess\_num = 1, 10, 10^2$, and 10^3.

Overall analysis. The results in Table 3 show the success rate of attacks across scenario #1∼#4 on the 4iQ and Naz.API datasets. The highest guessing success rate of our attack is 96.67% in scenario #1 on the 4iQ dataset and 50.86% in scenario #1 on the Naz.API dataset. The success rate in scenario #2 is the lowest because all leaked passwords are changed into random passwords. Scenario #1 has the highest success rate and #3 and #4 are lower, with differences of less than 7.5%. This highlights the importance of updating leaked passwords and the potential risks posed by users who ignore PM alerts [28], [40]. The scenario #3 and #4 have similar success rate which shows that an unstable network has limited effect on our attack. Across all scenarios, our attacks on the 4iQ dataset are more effective than those on Naz.API, due to the higher prevalence of popular passwords in 4iQ (12.74% on the 4iQ dataset, and 1.85% on the Naz.API dataset)[3].

Table 3 shows that, across different scenarios, the guessing success rates for leaked passwords remain similar, except in scenario #2. Notably, scenario #4 achieves a comparable guessing success rate despite its low identification rate (see details in Sec. 4.3). This is due to the credential connecting algorithm's capability to generate guessing candidates with as few as two matched credentials, even when the identification fails to recognize more than 80% of a PM user's credentials. This demonstrates the robustness of the credential connecting, emphasizing that recovery can remain effective even when identification is less successful.

Compare with the Basic. Compared to the basic attack, our attack achieves a guessing success rate improvement of up to 9.15% on the 4iQ dataset and 22.6% on the Naz.API dataset within 10^3 guesses. With only one guess, the improvement reaches as high as 30.43% on the Naz.API dataset. Given the lower proportion of popular passwords in the Naz.API dataset, greater improvements on the Naz.API dataset indicate that the effectiveness of our attack primarily stems from its ability to leverage users' previously leaked credentials. Given users' tendency to reuse or slightly modify old passwords, our attack poses a significant threat.

Compare with Li et al.'s attack [32]. On both the 4iQ and Naz.API datasets, the attack by Li et al. [32] is able to extract more unleaked passwords than ours. This advantage stems from two categories of users that our credential

3. The set of popular passwords are derived from prior studies [54], [56].

connecting approach fails to associate with the breach dataset: (1) PM users with only one leaked credential in the breach dataset. In this case, credential connecting cannot establish links between buckets to associate the user with the breach dataset[4]; (2) PM users who cannot be identified during the identification phase, such as those who issue very few queries or have only one credential in their vaults. The attack by Li et al. [32] can extract more leaked passwords from the above two categories of users, thereby enabling more effective guessing of unleaked passwords. For users without any leaked credentials, including previously leaked ones, both our attack and that of Li et al. [32] offer no advantage over the basic attack.

It is worth noting that our attack achieves a higher password guessing success rate than that of Li et al. [32] on the 4iQ dataset in scenarios #1, #3, and #4. This advantage arises from the increased presence of popular passwords, which amplifies the likelihood of collisions in the credential connecting algorithm. Specifically, for two distinct passwords, the probability that another two passwords share the same hash prefixes is approximately $\frac{1}{2^{40}}$. However, if one of the passwords is the same, this probability increases to about $\frac{1}{2^{20}}$. Popular passwords therefore lead to more hash prefix collisions, enabling the credential connecting algorithm to associate more leaked passwords. It is important to emphasize that the additional connected passwords in our attack are all from the breach dataset.

4.3. Further analysis for identification

To further illustrate the effect of the identification phase, we define metrics to evaluate the ability to identify users. Since the attacker may not know the exact number of users' credentials, they cannot precisely determine the length parameters for identifying. However, if an identified subsequence originates from one user, the attacker can use it to de-anonymize the user and proceed with an extraction phase. Based on this, we formally define *effective identification*.

Definition 1. *An identified subsequence V with j queries is considered a β-**effective identification** if at least $\lceil \beta \cdot j \rceil$ queries are from the same user, where β is a threshold.*

Next, we define *successful identification*. A successful identification goes beyond effective identification by identifying the majority of the users' credentials.

Definition 2. *For a PM user U with vault containing n credentials, denoted as $(u^1, pw^1), \ldots, (u^n, pw^n)$, an identified subsequence V is considered an α-**successful identification** if it covers at least $\lceil \alpha \cdot n \rceil$ of the user's credentials, , where α is a pre-defined threshold. In this case, the PM user U is deemed **successfully identified**.*

Based on the above definitions, we introduce two main metrics (i.e., %Usage and %Accuracy) and four auxiliary metrics to evaluate the identification phase in Table 4.

4. For simplicity, we omit the case where a PM user has two leaked credentials that fall into the same bucket due to identical hash prefixes. This scenario has a low probability $\frac{1}{2^{20}}$ of occurrence for one user.

TABLE 4: Metrics for evaluating the identification phase.

Metric	Description
#Identified	The number of identified subsequences.
#Effectiveness	The number of effective identifications achieved by the identification phase.
%Usage	The ratio $\frac{\#\text{Effectiveness}}{\#\text{Identified}}$, which indicates the precision of the identification phase. A smaller value for this metric suggests that the identification misidentifies more ordinary users as PM users.
#Total	The total number of PM users.
#Success	The number of PM users successfully identified by the identification phase.
%Accuracy	The ratio $\frac{\#\text{Success}}{\#\text{Total}}$, which measures the overall success rate of identification. A smaller value for this metric indicates either a low number of effective identifications or instances where identification misidentifies a single PM user as multiple users.

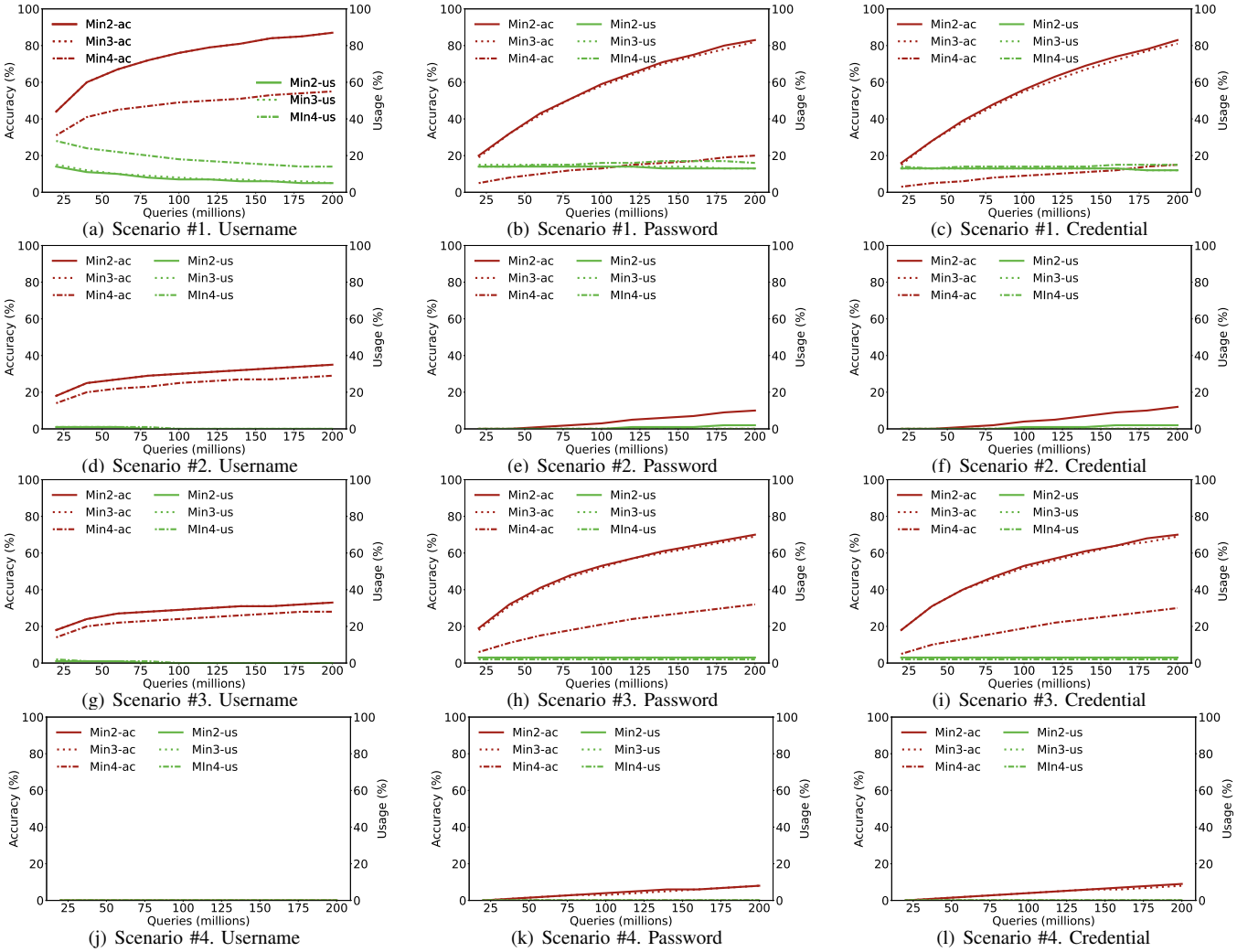
We evaluate the identification presented in Algo. 1 and 2, setting l from 10 to 20 and testing different minimum threshold parameters min_count with values of two, three, and four. The parameters α and β are set to 0.8 to define effective and successful identifications. Fig. 2 presents the %Accuracy and %Usage across different scenarios and bucket mechanisms for the Naz.API dataset. The results for the 4iQ dataset are provided in Fig. 3 in Appendix C.

In our simulation, PM users in the Naz.API dataset issue an average of 9.6 batch queries over 200 million queries, while users in the 4iQ dataset issue an average of 1.9 batch queries. Based on this, we adopt different pruning timelines and focus on different user groups. For the Naz.API dataset, we set the pruning threshold to 5 million queries and evaluate PM users with at least 10 batch queries. For the 4iQ dataset, the threshold is set to 20 million queries, and we focus on PM users with at least 4 batch queries.

The highest %Accuracy is achieved in scenario #1 using the username-based bucket with $min\_count = 2$, reaching 87.59% in the Naz.API dataset. The highest %Usage is observed in scenario #1 with the password-based bucket and $min\_count = 4$, achieving 73.16%. Detailed analyses are provided below. The impact of the parameter w is discussed in Appendix C, while potential optimizations of identification are explored in Appendix D.

The minimal threshold parameters. The results indicate that lower min_count values result in higher %Accuracy, with $min\_count = 2$ achieving the highest %Accuracy across all scenarios. This occurs because sequences that fail to accumulate enough appearances to exceed the threshold before pruning are less likely to meet the threshold in subsequent queries. However, lower min_count values do not yield better %Usage, because lower min_count increases both #Effectiveness and #Identified, meaning that it identifies more correct queries but also incorrect ones.

Different scenarios and bucket mechanisms. Scenario #1 achieves the highest %Accuracy because the batch queries sent by the PM users are received by the server in exactly the same order, allowing the algorithm to fully capture these sequences. In Scenario #2, %Accuracy decreases: password-based bucket mechanism drops the most to 10.99%, username-based bucket mechanism drops less to 35.20%, and credential-based bucket decreases to



(a) Scenario #1. Username (b) Scenario #1. Password (c) Scenario #1. Credential

(d) Scenario #2. Username (e) Scenario #2. Password (f) Scenario #2. Credential

(g) Scenario #3. Username (h) Scenario #3. Password (i) Scenario #3. Credential

(j) Scenario #4. Username (k) Scenario #4. Password (l) Scenario #4. Credential

Figure 2: Experiments on subsequence identification with lengths ranging from 10 to 20 and min_count set to two, three, and four. The results report %Accuracy and %Usage across scenarios #1 to #4 under three different bucket mechanisms: username-based, password-based, and credential-based. The evaluation is conducted on the Naz.API dataset.

12.10% for Naz.API dataset. This variation occurs because users change leaked passwords but typically do not change usernames, making the username-based bucket less affected. The credential-based bucket mechanism lies in between, as a leaked password does not necessarily imply a leaked credential. For example, if the credential (`abc123@abc.com`, `loveu4ever`) is leaked but (`def456@def.com`, `loveu4ever`) is not, the latter will not trigger an alert, and is thus likely to remain unchanged.

In Scenario #3, all bucket mechanisms show an increase in %Accuracy, highlighting that the practice of changing leaked passwords significantly strengthens resistance to identification attacks. This resistance is more effective for password-based bucket mechanisms and less impactful for username-based bucket mechanisms. In Scenario #4, both %Accuracy and %Usage decrease sharply. This is due to increased interceptions in PM users' queries, which reduce the effectiveness of query recognition.

4.4. Further analysis for credential connecting

After identification, credential connecting uses the identified subsequences to connect credentials in breach dataset. In this section, we evaluate the credential connecting algorithm in Algo. 3. The parameter $min\_overlap$ represents the number of overlapping credentials between the identified subsequences and the credentials of users in the breach dataset. In our experiments, we evaluate $min\_overlap = 2, 3, 4, 5$ to analyze how many credentials of each users in breach dataset credential connecting can connect. For efficiency considerations, we only use users with at least 10 credentials in the breach dataset for connection.

We evaluate three metrics: connected rate, popular rate and success rate. The connected rate measures the proportion of connected subsequences among all identified subsequences. This metric reflects the extent to which identified subsequences can be utilized in credential connecting. The

popular rate measures the proportion of popular passwords among all connected passwords, illustrating the role of popular passwords in connections.

The success rate measures the proportion of successfully connected users among the PM users who are expected to be connected. A user is considered *successfully connected* if more than $min\_overlap = 2, 3, 4, 5$ of their credentials are matched through credential connecting. PM users who are expected to be connected must also have more than $min\_overlap = 2, 3, 4, 5$ of their credentials matching those found in the breach dataset.

The detailed results are presented in Table 8 in Appendix C. In summary, our evaluation shows that the habit of credential reuse does not facilitate credential connecting, as reused credentials are mapped to the same bucket. This is particularly evident in the username-based bucket mechanism, which exhibits a low connected rate because users tend to reuse the same usernames (e.g., email addresses). Moreover, previously used popular passwords may inadvertently increase both the connected and success rates. Even after users change their leaked popular passwords, queries from other users containing these passwords may co-occur within the same batch query, allowing the attacker to link the user back to the breach dataset.

5. Query Patterns of Password Managers

In this section, we identify the de facto query patterns of password managers (PMs) in the wild. We focus on the following three key research questions (RQs):

RQ1: *What conditions trigger password managers to automatically check users' credentials?*

RQ2: *What are the query patterns of password managers within a single batch query to C3 services?*

RQ3: *During a batch query, do password managers leak additional information, such as cookies, that could assist attackers in identifying users more effectively?*

The selection of PMs and the methodology for collecting query patterns are described in Appendices E and F.

5.1. Collection results

Among the 14 PMs we analyzed, 8 explicitly integrate the Have I Been Pwned (HIBP) API [24] to offer C3 services. Chrome PM utilizes Google Password Checkup (GPC) [49], Sticky Password integrates with Arc, and LastPass relies on Enzoic. The remaining three PMs operate their own C3 services by maintaining proprietary breach datasets.

For **RQ1**, we find that the C3 functionality can be triggered either *actively* by users or *passively* by the PMs when users manage their credentials. Among the 14 PMs, 6 support active checking, which is triggered by manually clicking a check button. 11 PMs support passive checking, which is triggered under three conditions: (1) Triggered after adding or changing credentials: Dashlane, RoboForm, Avira PM, Enpass, 1Password, and KeePass automatically check all credentials when users add or change credentials, while Bitwarden triggers a check when users register or modify master passwords; (2) Triggered upon user login: Chrome PM and KeePass detect the breach status of all accounts, while Dashlane checks the master password at each login; (3) Triggered for specific credential: Microsoft's PM checks a credential whenever it is saved, autofilled, or added for monitoring, and Mozilla Monitor detects a specific email address when it is added for monitoring.

Furthermore, Chrome PM, Microsoft's PM, Mozilla Monitor, KeePass, and Dashlane (for checkups with email addresses) require users to manually enable passive checking, whereas in other cases it is enabled by default. Specifically, Avira PM and Dashlane (for checkups with passwords) do not allow passive checking to be disabled.

The results in Table 5 show that current PMs implement various passive trigger conditions. However, as shown in Sec. 4.3, attackers can more easily identify batch queries when the number of queried credentials is large. Therefore, the passive triggers with frequent user actions can significantly strengthen the effectiveness of identification. Specifically, for batch queries triggered by autofill functionality, if the C3 queries are tied to the use of a specific credential and that credential corresponds to a popular website, batch queries will occur more frequently. This, in turn, makes it easier for the attacker to perform accurate identification.

For **RQ2**, PMs use different query types and patterns. Among the 14 PMs, 9 support password-based C3 queries, 5 support username-based queries, and only 2 use full credentials for detection. For each batch query, all PMs send queries in a fixed order. Specifically, LastPass sends queries alphabetically by email address, Dashlane follows the order in which email addresses are added, and the others adhere to the order in which passwords are stored.

Additionally, Table 5 shows that 9/14 PMs repeatedly query the same credentials within a single batch query. Given the widespread practice of password reuse [18], [44], [51], [52], [53], this query pattern makes identification easier (see details in Sec. 3.1). The Chrome PM caches hash prefixes in the browser if it detects that a credential has been leaked [49]. This prevents repeated queries for breached credentials, but unbreached credentials are still queried multiple times. LastPass, Dashlane, and NordPass check the user's email address and require the user to confirm the email address via a verification email before proceeding with the check. During this process, they avoid sending duplicate verification emails.

LastPass and NordPass claim to monitor users' emails continuously. Dashlane checks users' master passwords and stored email addresses daily, while Mozilla Monitor checks users' credentials monthly. These periodic query patterns make it easier for attackers to identify and monitor PM users, as attackers can leverage the timing of these queries to track batch queries and make identification straightforward.

For **RQ3**, Table 5 shows that 8 out of 14 PMs embed cookies directly in HTTPS traffic. These cookies make it easier for attackers to confirm whether multiple queries originate from the same user. The captured packets show that PMs integrated with HIBP use GET requests to query

TABLE 5: Overview of C3 query patterns in password managers.[†]

Password manager@Version	C3 service integration	Credentials checked[*]	Trigger condition[‡] Type	O1	O2	O3	O4	Owner proof[§]	Cookie usage	Fixed order	Periodic queries	Duplicate queries[¶]	Issue disclosure[⋆]
Chrome PM@131.0.6778.265	GPC	MC	✋⚙^{c}	✓	×	✓	×	×	×	✓	×	✓	●
Microsoft's PM@131.0.2903.146	Self-support	MC	✋⚙^{c}	✓	×	×	✓	×	×	✓	×	✓	●
Mozilla Monitor@134.0.1	HIBP-UN	SE	⚙^{c}	×	✓	×	×	✓	✓	-	✓	-	○
LastPass@4.138.2	Enzoic	ME	✋	✓	×	×	×	✓	✓	✓	×	×	●
Sticky Password@8.8.6.1987	Arc	MP	✋	✓	×	×	×	×	✓	✓	×	✓	●
Bitwarden@2024.12.4	HIBP-UN	SE	✋	✓	×	×	×	×	×	-	×	-	○
	HIBP-PW	MP	✋⚙^{o}	✓	✓	×	×	×	✓	✓	×	✓	●
KeePass@2.57.1	HIBP-PW	MP	⚙^{c}	×	✓	✓	×	×	✓	✓	×	✓	◐
RoboForm@9.6.12.0	HIBP-PW	MP	⚙^{o}	×	✓	×	×	×	✓	✓	×	✓	◐
Enpass@6.11.2	HIBP-PW	MP	⚙^{o}	×	✓	×	×	×	×	✓	×	✓	●
1Password@8.10.56-22	HIBP-PW	MP	⚙^{o}	×	✓	×	×	×	✓	✓	×	✓	◐
Avira PM@2.21.0.4979	HIBP-PW	MP	⚙^{d}	×	✓	×	×	×	×	✓	×	✓	◐
SafeInCloud @24.14.1	HIBP-PW	MP	✋	✓	×	×	×	×	×	✓	×	✓	●
NordPass @5.24.15	Self-support	ME	⚙^{o}	×	✓	×	×	✓	✓	✓	×	×	○
Dashlane@6.2448.0	Self-support	ME	⚙^{c}	×	✓	×	×	✓	×	✓	✓	×	○
		SP	⚙^{d}	×	×	✓	×	✓	×	-	✓	-	○

[†] ✓: Yes; ×: No; "-": Not considered; "HIBP-UN": HIBP using the username query type; "HIBP-PW": HIBP using the password query type.
[*] MC: Multiple credential pairs; SE: Single email address; ME: Multiple email addresses; SP: Single password; MP: Multiple passwords.
[‡] ✋: Actively triggered by users; ⚙^{o}: Passively triggered by the password manager, enabled by default; ⚙^{c}: Passively triggered by the password manager, disabled by default; ⚙^{d}: Passively triggered by the password manager, enabled by default and cannot be disabled; O1: Triggered by manually clicking a check button; O2: Triggered after adding or changing credentials; O3:Triggered upon user login; O4: Triggered after autofilling.
[⋆] ●: We have disclosed the potential query pattern leakage to the password manager and received feedback; ◐: We have disclosed the potential query pattern leakage to the password manager but not received feedback; ○: We do not disclose vulnerabilities to PMs that query only one credential at a time, since identification in credential extraction attacks would be ineffective. In addition, we did not disclose the issue to NordPass and Dashlane regarding the checking of multiple email addresses, as they do not consider this a feature requiring protection and perform the checkup using plaintext emails.
[§]: "Owner proof" denotes whether the PM requires proof that the queried credential belongs to the user.
[¶]: "Duplicate queries" denotes whether a PM re-queries the same credential in a batch (for PMs that allow only one credential per batch, this is shown as "-").

the API, with parameters containing the first 20 bits of the SHA1 hash of the queried password. These URLs are stored in the browser history, potentially exposing sensitive hash prefixes to attackers locally.

Remark. We note that IP addresses have a similar effect to cookies. IP addresses can help determine whether a single batch query originates from the same user. However, IP addresses are easily affected by changes in device, location, or VPN usage. In contrast, cookies typically remain unchanged for a user over a certain period, and the hash prefixes of batch queries tend to remain stable, as a user's vault usually does not change significantly over a period.

Disclosure. The results of the disclosure are summarized in Table 5. We have disclosed all of our findings to the 10 affected PM vendors and provided recommendations to mitigate the vulnerabilities. As of June 5, 2025, we have received responses from seven vendors.

Among the vendors, we disclosed the issue to all these vendors on January 17, 2025. The SafeInCloud support team acknowledged our findings and implemented our recommendations, stating that they "take these concerns seriously and will evaluate your thoughtful mitigation suggestions" on January 19. Enpass reviewed the vulnerabilities we identified and responded that they "will follow up with an update as soon as possible" on January 23. Bitwarden also acknowledged our disclosure invited us to participate in their bug bounty program. Bitwarden responded that they have "had these issues reported in the past, and have had some lengthy discussions internally" on February 15.

On January 18, Sticky Password explained that they adopt an atypical C3 protocol, and therefore our analysis does not directly apply to their implementation. However, the C3 protocol they rely on, Arc, also employs a deterministic bucket mechanism. As a result, we engaged in further technical discussions with the Sticky Password team to better understand their protocol and security considerations. On February 17, Chrome PM claimed that its username-based bucket mechanism is secure against our attack. However, as demonstrated in Sec. 4.4, the identification phase still has an observable effect on their mechanism. On February 19, Microsoft PM indicated that, since they operated their own C3 service, they did not consider the attack severe enough to require immediate remediation. LastPass invited us to their Bugcrowd program on February 14, and we still engage in further technical discussions with them.

6. Discussion

Our evaluation demonstrates that the query patterns of PMs can provide attackers with a significant advantage in guessing users' credentials. Given that current privacy-preserving protocols cannot handle the scale of data required by C3 services, which is on the order of billions credentials, we leave the design of a query-pattern-leakage-free C3 protocol as future work. In this work, we focus on mitigating query pattern leakage from the PM side and offer promising recommendations. Additionally, we note that the identification phase of our attacks may pose additional privacy threats, and we discuss other potential risks in detail.

Mitigating query pattern leakage. Based on observation 1 in Sec. 1, one common query pattern of PMs is querying all credentials of a user, which transforms the identification task into a subsequence-identification problem. However, identification becomes ineffective for batch queries that contain only a single credential, since such queries reveal only a credential prefix and cannot be linked to adjacent

queries. Therefore, we recommend that PMs query only one credential at a time. For example, when a user adds or updates a credential, the PM should query only that specific credential. In addition, PMs can associate a timestamp with each credential and query the C3 service only when the credential has not been checked for an extended period. Credential checking based on cookies should be avoided, as it may facilitate identification attacks.

According to our evaluation of scenario #2 (see Sec. 4), updating leaked passwords is crucial. Popular PMs encourage users to change their leaked passwords to randomly generated ones. However, users may still ignore these alerts [40]. Therefore, we recommend that PMs store metadata indicating the results of password compromise checks. If a credential is identified as leaked, it should not be queried again until it has been updated. Additionally, supporting automatic password updates may further encourage users to replace compromised credentials promptly.

As shown in scenario #2 of Table 3, timely updating leaked passwords is an effective mitigation against credential extraction attacks for PM users employing password-based bucketization in C3, particularly when updated with strong and random passwords. Users can also disable passive compromise checking and avoid running checkups on the entire vault at once to further reduce the risk of identification by a semi-honest C3 server. In addition, for password managers that do not employ C3 service with password-based bucketization, users may adopt one-time masked emails [23] to protect their usernames by creating a unique email address for each credential.

The impact of identification. After identification, the attacker gets a number of identified subsequences. This enables the attacker to trace the query behavior of PM users. According to the "trigger condition" column in Table 5, batch queries issued by PMs can be triggered by user actions on their vaults. As a result, the attacker can monitor specific PM users to infer their vault contents and activity patterns.

Specifically, when two batch queries differ by only one internal query, the attacker can infer that the later batch query may have been triggered by a credential update in the vault. The differing query likely corresponds to the newly modified credential, which may become a new target for password extraction. When two batch queries are identical, the attacker can infer that the user may have logged into the PM or used it to autofill credentials on websites. All of these passive automatic checks introduce new privacy risks.

Other potential threats. In our work, we focus on the query patterns of PMs. However, ordinary users may exhibit distinct query behaviors. These users primarily access C3 services through websites and manually enter their credentials for checking. This usage pattern limits their ability to perform frequent and comprehensive checks of their entire vault. Nonetheless, users may still display habits that are identifiable, such as avoiding the simultaneous checking of duplicate credentials. As a result, attackers could potentially apply general machine learning or deep learning algorithms to extract query patterns and identify individual ordinary users. We leave the investigation of their query patterns and corresponding identification methods as future work.

As shown in Sec. 4, updating leaked passwords has a significant impact on the effectiveness of the query pattern leakage attack. Therefore, a malicious C3 server could deviate from the original protocol and falsely inform users that certain leaked passwords are safe. This scenario further underscores the importance of designing a verifiable C3 protocol to ensure strong security guarantees in the presence of potentially malicious servers.

7. Conclusion

We have, for the first time, explored the query pattern leakage of C3 services and proposed a novel attack targeting the query patterns of password managers (PMs). Extensive empirical experiments demonstrate that these patterns can be exploited by honest-but-curious C3 servers to effectively identify users and recover their credentials. This finding reveals that even a few bits of leakage in C3 protocols can pose greater risks than previously believed [32], [42].

We further showed that 10 out of 14 popular PMs are vulnerable to such attacks.We have shared our findings with the affected PM vendors and provided recommendations to mitigate these risks. Given the growing adoption of password managers, we believe that our work offers important insights for enhancing the security of users' credentials.

Acknowledgments

The authors are grateful to the shepherd and anonymous reviewers for their invaluable comments. Ding Wang is the corresponding author. This research was in part supported by the National Natural Science Foundation of China under Grants Nos. 62222208 and 62572259, and by the Natural Science Foundation of Tianjin, China under Grant No. 25JCJQJC00230, and by the Fundamental Research Funds for the Central Universities, Nankai University (Grant No. 63243154). See more at http://bit.ly/48x9Cw2 .

References

[1] *Password managers: using browsers and apps to safely store your passwords*, Dec. 2021, https://www.ncsc.gov.uk/collection/top-tips-for-staying-secure-online/password-managers.

[2] *Why the password isn't dead quite yet? Everyone hates the old ways of authentication. But change comes with its own drawbacks.*, Jul. 2021, https://tinyurl.com/56xnf3yt.

[3] *NIST SP 800-63: Digital Identity Guidelines Frequently Asked Questions.*, Mar. 2022, https://pages.nist.gov/800-63-FAQ/.

[4] *The Password Isn't Dead Yet. You Need a Hardware Key*, 2022, https://www.wired.com/story/hardware-security-key-passwords-passkeys/.

[5] *Inside the Massive Naz.API Credential Stuffing List*, 2024, https://tinyurl.com/44e99bba.

[6] *Have I Been Pwned Usage Statistics*, 2025, https://trends.builtwith.com/widgets/Have-I-Been-Pwned.

[7] 1Password, *Protect your accounts with strong, unique passwords.*, 2025, https://1password.com/haveibeenpwned/.

[8] A. Ahmed, *2024 Study: 36% Use Password Managers, 79% Opt for Free, and Google Leads with 32% Adoption*, Nov. 2024, https://www.digitalinformationworld.com/2024/11/2024-study-36-use-password-managers-79.html.

[9] S. A. Al-Roomi and F. Li, "A large-scale measurement of website login policies," in *Proc. USENIX SEC 2023*, pp. 2061–2078.

[10] J. Ali, *Validating Leaked Passwords with k-Anonymity*, 2018, https://blog.cloudflare.com/validating-leaked-passwords-with-k-anonymity/.

[11] S. Alroomi and F. Li, "Measuring website password creation policies at scale," in *Proc. CCS 2023*, pp. 3108–3122.

[12] S. Amft, S. Höltervennhoff, N. Huaman, Y. Acar, and S. Fahl, ""would you give the same priority to the bank and a game? I do not!" exploring credential management strategies and obstacles during password manager setup," in *Proc. SOUPS 2023*, pp. 171–190.

[13] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh, "Kamouflage: Loss-resistant password management," in *Proc. ESORICS 2010*.

[14] J. Bonneau, C. Herley, P. van Oorschot, and F. Stajano, "Passwords and the evolution of imperfect authentication," *Commun. ACM*, vol. 58, no. 7, pp. 78–87, 2015.

[15] J. Bonneau, C. Herley, P. C. Van Oorschot, and F. Stajano, "The quest to replace passwords: A framework for comparative evaluation of web authentication schemes," in *Proc. IEEE S&P 2012*, pp. 553–567.

[16] J. Casal, *1.4 Billion Clear Text Credentials Discovered in a Single Database*, 2017, https://tinyurl.com/ux3pspze.

[17] R. Chatterjee, J. Bonneau, A. Juels, and T. Ristenpart, "Cracking-resistant password vaults using natural language encoders," in *Proc. IEEE S&P 2015*, pp. 481–498.

[18] A. Das, J. Bonneau, M. Caesar, N. Borisov, and X. Wang, "The tangled web of password reuse," in *Proc. NDSS*, 2014, pp. 1–15.

[19] Y. Duan, D. Wang, and Y. Fu, "Security analysis of master-password-protected password management protocols," in *Proc. IEEE S&P 2025*.

[20] Y. Fu and D. Wang, "Leaky autofill: An empirical study on the privacy threat of password managers' autofill functionality," in *Proc. ACSAC 2024*, pp. 1–16.

[21] P. Gasti and K. B. Rasmussen, "On the security of password manager database formats," in *Proc. ESORICS 2012*, pp. 770–787.

[22] Y. Gupta, *Are passwords dead? Experts weigh in on World Password Day*, May 2024, https://dynamicbusiness.com/featured/are-passwords-dead-experts-weigh-in-on-world-password-day.html.

[23] M. Hanley, *Protect your privacy with 1Password and Fastmail*, 2021, https://blog.1password.com/fastmail-masked-email/.

[24] Have I Been Pwned, *Who's been pwned*, 2025, https://haveibeenpwned.com/PwnedWebsites,.

[25] T. Hunt, *Seized Genesis Market Data is Now Searchable in Have I Been Pwned, Courtesy of the FBI and "Operation Cookie Monster"*, 2023, https://tinyurl.com/v3bvkvez.

[26] A. Hutchinson, C. W. Munyendo, A. J. Aviv, and P. Mayer, "An analysis of password managers' password checkup tools," in *Proc. CHI 2024*, pp. 38:1–38:7.

[27] IBM Security, *Cost of a Data Breach Report 2021*, 2021, https://info.techdata.com/rs/946-OMQ-360/images/Cost_of_a_Data_Breach_Report_2021.PDF.

[28] E. Kablo, K. Kader, and P. A. Cabarcos, ""i'm actually going to go and change these passwords": Analyzing the usability of credential audit interfaces in password managers," in *Proc. CHI Extended Abstracts 2024*, pp. 2:1–2:13.

[29] LastPass, *What is dark web monitoring?*, 2024, https://support.lastpass.com/s/document-item?language=en_US&bundleId=lastpass&topicId=LastPass/c_lastpass_faqs_users_about_dark_web_monitoring.html&_LANG=enus.

[30] K. Lauter, *Password Monitor: Safeguarding passwords in Microsoft Edge*, 2021, https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/.

[31] J. Li, Y. Liu, and S. Wu, "Pipa: Privacy-preserving password checkup via homomorphic encryption," in *Proc. ACM AsiaCCS 2021*.

[32] L. Li, B. Pal, J. Ali, N. Sullivan, R. Chatterjee, and T. Ristenpart, "Protocols for checking compromised credentials," in *Proc. ACM CCS 2019*, pp. 1387–1403.

[33] X. Lin, P. Ilia, and J. Polakis, "Fill in the blanks: Empirical analysis of the privacy threats of browser form autofill," in *Proc. ACM CCS 2020*, pp. 507–519.

[34] P. Mayer, C. W. Munyendo, M. L. Mazurek, and A. J. Aviv, "Why users (don't) use password managers at a large educational institution," in *Proc. USENIX SEC 2022*, pp. 1849–1866.

[35] C. W. Munyendo, P. Mayer, and A. J. Aviv, ""I just stopped using one and started using the other": Motivations, techniques, and challenges when switching password managers," in *Proc. CCS 2023*.

[36] NordPass, *People have around 170 passwords on average, study shows*, May 2024, https://tinyurl.com/ycy4tr8y.

[37] S. Oesch, A. Gautam, and S. Ruoti, "The emperor's new autofill framework: A security analysis of autofill on ios and android," in *Proc. ACM ACSAC 2021*, pp. 996–1010.

[38] S. Oesch and S. Ruoti, "That was then, this is now: A security evaluation of password generation, storage, and autofill in browser-based password managers," in *Proc. USENIX SEC 2020*.

[39] S. Oesch, S. Ruoti, J. Simmons, and A. Gautam, ""It basically started using me:" an observational study of password manager usage," in *Proc. CHI 2022*, pp. 1–23.

[40] S. Oh, H. Baek, J. H. Huh, T. Kim, W. Jeon, I. Oakley, and H. Kim, "Understanding and improving user adoption and security awareness in password checkup services," in *Proc. CHI 2025*, pp. 386:1–386:32.

[41] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart, "Beyond credential stuffing: Password similarity models using neural networks," in *Proc. IEEE S&P 2019*, pp. 417–434.

[42] B. Pal, M. Islam, M. S. Bohuk, N. Sullivan, L. Valenta, T. Whalen, C. A. Wood, T. Ristenpart, and R. Chatterjee, "Might I get pwned: A second generation compromised credential checking service," in *Proc. USENIX SEC 2022*, pp. 1831–1848.

[43] D. Pasquini, D. Francati, G. Ateniese, and E. M. Kornaropoulos, "Breach extraction attacks: Exposing and addressing the leakage in second generation compromised credential checking services," in *Proc. IEEE S&P 2024*, pp. 1405–1423.

[44] S. Pearman, J. Thomas, P. E. Naeini, H. Habib, L. Bauer, N. Christin, L. F. Cranor, S. Egelman, and A. Forget, "Let's go in for a closer look: Observing passwords in their natural habitat," in *Proc. ACM CCS 2017*, pp. 295–310.

[45] G. V. Research, *Password Management Market Size & Trends*, Nov. 2024, https://www.grandviewresearch.com/industry-analysis/password-management-market.

[46] Security.org Team, *2024 Password Manager Industry Report and Statistics*, Nov. 2024, https://www.security.org/digital-safety/password-manager-annual-report/.

[47] D. Silver, S. Jana, D. Boneh, E. Y. Chen, and C. Jackson, "Password managers: Attacks and defenses," in *Proc. USENIX SEC 2014*.

[48] B. Stock and M. Johns, "Protecting users against XSS-based password manager abuse," in *Proc. ACM AsiaCCS 2014*, pp. 183–194.

[49] K. Thomas, J. Pullman, K. Yeo, A. Raghunathan, P. G. Kelley, L. Invernizzi, B. Benko, T. Pietraszek, S. Patel, D. Boneh *et al.*, "Protecting accounts from credential stuffing with password breach alerting," in *Proc. USENIX SEC 2019*, pp. 1556–1571.

[50] L. Valenta, C. D. Rubin, and C. Wood, *Privacy-Preserving Compromised Credential Checking*, 2021, https://blog.cloudflare.com/privacy-preserving-compromised-credential-checking/.

[51] C. Wang, S. T. Jan, H. Hu, D. Bossart, and G. Wang, "The next domino to fall: Empirical analysis of user passwords across online services," in *Proc. CODASPY 2018*, pp. 196–203.

[52] D. Wang, D. He, H. Cheng, and P. Wang, "fuzzyPSM: A new password strength meter using fuzzy probabilistic context-free grammars," in *Proc. IEEE/IFIP DSN*, 2016, pp. 595–606.

[53] D. Wang, Z. Zhang, P. Wang, J. Yan, and X. Huang, "Targeted online password guessing: An underestimated threat," in *Proc. ACM CCS 2016*, pp. 1242–1254.

[54] D. Wang, Y. Zou, Y.-A. Xiao, S. Ma, and X. Chen, "PASS2EDIT: A multi-step generative model for guessing edited passwords," in *Proc. USENIX SEC 2023*, pp. 1–18.

[55] Wikipedia, *List of data breaches*, 2025, https://en.wikipedia.org/wiki/List_of_data_breaches.

[56] K. Xiu and D. Wang, "Pointerguess: Targeted password guessing model using pointer mechanism," in *Proc. USENIX SEC 2024*.

[57] A. Zaharia, *Have I Been Hacked? How To Recognize & Recover From a Hack*, 2024, https://www.aura.com/learn/have-i-been-hacked.

[58] R. Zhao and C. Yue, "All your browser-saved passwords could belong to us: A security analysis and a cloud-based new design," in *Proc. CODASPY 2013*, pp. 333–340.

[59] R. Zhao, C. Yue, and K. Sun, "Vulnerability and risk analysis of two commercial browser and cloud based password managers," *ASE Science Journal*, vol. 1, no. 4, pp. 1–15, 2013.

[60] S. Zibaei, D. R. Malapaya, B. Mercier, A. Salehi-Abari, and J. Thorpe, "Do password managers nudge secure (random) passwords?" in *Proc. SOUPS 2022*, pp. 581–597.

Appendix A. Theorem 1 and its proof

The probability of observing repeated subsequences from ordinary users is small. This is because ordinary users typically possess at most a few hundred credentials [36], whereas C3 services process at least millions of queries one day [49], making such repetition statistically unlikely.

Theorem 1. *Let $\mathcal{D}$ be a distribution over a credential space. Let $Q_1, \cdots, Q_N$ be i.i.d. draws from $\mathcal{D}$. Let $\mathcal{H} : \{0,1\}^* \to \{0,1\}^k$ be a random oracle independent of the Q_i's. Fix a subsequence $S = (q_1, \cdots, q_l)$ and define $p = \Pr(q_1) \cdot \Pr(q_2) \cdot \cdots \cdot \Pr(q_l)$. For each start $t \in \{1, \cdots, N-l+1\}$, define $I_t = \mathbf{1}\{(Q_t, \cdots, Q_{t-l+1}) = S\}$ and $Y = \sum_{t=1}^{N-l+1} I_t$. For $(1 \le d < l)$, define $\beta_d = \prod_{i=l-d+1}^{l} \Pr(q_i)$ if $(q_1, \ldots, q_{l-d}) = (q_{d+1}, \ldots, q_l)$, otherwise $\beta_d = 0$. Then Y is an l-locally dependent sum with mean $\lambda = (N-l+1)p$, and the total variation distance to $\mathrm{Poisson}(\lambda)$ obeys $d_{\mathrm{TV}}\big(\mathcal{L}(Y), \mathrm{Poisson}(\lambda)\big) \le \varepsilon_{\mathrm{TV}}$ where $\varepsilon_{\mathrm{TV}} = \min\{1, \frac{1}{\lambda}\}\Big((N-l+1)\,p^2 + 2\sum_{d=1}^{l-1}(N-l+1-d)\,p\,\beta_d\Big)$. Consequently, for every integer $m \ge 0$,*

$$\max\big\{0,\ \Pr(\mathrm{Poisson}(\lambda) \ge m) - \varepsilon_{\mathrm{TV}}\big\} \le \Pr(Y \ge m) \le \min\big\{1,\ \Pr(\mathrm{Poisson}(\lambda) \ge m) + \varepsilon_{\mathrm{TV}}\big\}.$$

The same bounds hold when matching in the hash space of a k-bit random oracle, up to an additional additive collision term $\mathcal{O}(l/2^k)$.

Proof. Define I_t as above and note that $\mathbb{E}[I_t] = p$, hence $\mathbb{E}[Y] = \lambda = (N-l+1)p$. If $|t-s| \ge l$, the windows are disjoint and I_t and I_s are independent. When $1 \le d = |t-s| \le l-1$, both events $I_t = I_s = 1$ can occur only if S has a border of length $l-d$; in that case $\Pr(I_t = I_s = 1) = p\beta_d$, and otherwise this probability is zero. Thus Y is a sum of Bernoulli indicators with l-local dependence. Applying the standard Chen-Stein local-dependence bound yields $d_{\mathrm{TV}}\big(\mathcal{L}(Y), \mathrm{Poisson}(\lambda)\big) \le \min\{1, 1/\lambda\}\Big(\sum_t \mathbb{E}[I_t]^2 + \sum_t \sum_{1 \le d < l} \mathbb{E}[I_t I_{t+d}]\Big)$, which is exactly $\varepsilon_{\mathrm{TV}}$ after substituting $\sum_t \mathbb{E}[I_t]^2 = (N-l+1)p^2$ and $\sum_t \sum_{1 \le d < l} \mathbb{E}[I_t I_{t+d}] = 2\sum_{d=1}^{l-1}(N-l+1-d)\,p\,\beta_d$.

By the definition of total variation distance, we have $|\Pr(Y \in A) - \Pr(\mathrm{Poisson}(\lambda) \in A)| \le \varepsilon_{\mathrm{TV}}$, and taking $A = \{m, m+1, \ldots\}$ gives the stated two-sided tail bounds. The random-oracle model is used to argue that equality in the hash space coincides with equality in the input space up to at most $\mathcal{O}(l/2^k)$ collisions by a union bound. □

When $\sum_{d=1}^{l-1} \beta_d$ is small (e.g., uniform k-bit random oracle with $\beta_d \le 2^{-kd}$), the error $\varepsilon_{\mathrm{TV}}$ is extremely small and the Poisson tail is numerically accurate. In our setting, l is far smaller than N: users typically hold about 80-168 credentials in practice, while C3 servers process on the order of one billion queries per day. For example, consider $N = 2,000,000$ queries and a subsequence length of $l = 80$, where each query is modeled as a 20-bit random-oracle output. Under a uniform distribution, the probability that a fixed subsequence S matches at a given starting position is $p(S) = 2^{-1600} \approx 10^{-481}$. With $N - l + 1 \approx 2 \times 10^6$ possible positions, the probability that S appears at least twice is approximately 2×10^{-950}, which is negligible.

Appendix B. Our datasets

In our evaluation, we use the 4iQ password leaks collection [16] and Naz.API [5]. 4iQ breach dataset contains about 1.4 billion username-password pair and has been widely used in prior research for empirical experiments [32], [41], [42], [43]. Naz.API breach dataset origin files contain over 100GB credentials and after removing junk data (e.g., duplicated files) remains about 300 millions credentials. We preprocess both password datasets following the same procedure as previous studies [32], [41], [42], [43]: removing passwords with more than 30 characters or non-ASCII characters, applying the mixed method [41] to merge usernames, and excluding users with more than 1,000 credentials, as they are unlikely to represent real individuals.

We partition the dataset D into two parts: 90% of the data is used to simulate the leaked dataset S for the C3 service, while the remaining 10% forms the test dataset T, which is used to generate queries to the C3 service. From the test dataset T, users with more than one password are categorized as PM users P, while the rest are classified as ordinary users O who do not use a PM. The statistical details of the dataset splits are presented in Table 6.

TABLE 6: Statistical information of 4iQ and Naz.API. †

Dataset	Statistic	D	S	T	O	P
4iQ	Usernames	950,495	875,435	125,434	119,249	6,184
	Passwords	435,088	403,144	69,735	64,507	7,448
	Unique u-pw	1,294,290	1,009,160	129,181	119,247	13,778
	Total u-pw	1,334,363	1,200,931	133,431	119,249	14,181
Naz	Usernames	80,824	75,290	14,223	11,852	2,565
	Passwords	62,043	58,092	12,230	9,288	4,267
	Unique u-pw	130,542	120,268	18,186	12,725	5,461
	Total u-pw	208,539	187,175	20,793	12,725	8,068

†: All statistical information includes the number (in thousands) of unique usernames, unique passwords, unique username-password pairs, and total username-password pairs. Here, D denotes the entire dataset (4iQ and Naz.API), S represents the simulated breach dataset, T the simulated test dataset, O the ordinary users within T, and P the PM users within T.

Remark. Note that while the actual adoption rate of password managers is approximately 30% [46], our simulation yields rates of 4.8% in the 4iQ dataset and 18% in the Naz.API dataset. This deviation is unavoidable because, to the best of our knowledge, there are no public datasets containing password information specific to PM users. However, it does not negatively impact our results, as the effectiveness of our attacks primarily depends on the clustering and repetitive characteristics of PMs' queries, as well as users' password reuse habits [41], [54]. A higher adoption rate would only indicate that the threat is even more severe.

Appendix C. Supplementary experiment results

Credential extraction attacks for username-based and credential-based bucket mechanisms. Given that there are no well-established guessing models for username-based and credential-based mechanisms, we adopt heuristic methods and report only the success rate under the constraint of a single guess. Under the username-based bucket mechanism, all email addresses sharing the same prefix are treated as belonging to the same user. This enables an attacker to easily generate additional candidate usernames for an identified user by altering the email domain. In our evaluation, we construct the guess list using all email domains available in the breach dataset. As shown in Table 7, our attack successfully extracts between 52.35% and 60.13% of usernames in the 4iQ dataset, and between 30.67% and 41.07% in the Naz.API dataset.

TABLE 7: Guessing success rates under username-based and credential-based bucketizations for PM users.‡

Bucket	#1	#2	#3	#4
User	52.35/41.07%	53.97/33.29%	54.84/32.03%	60.13/30.67%
Cred	1.62/32.18%	0.00/0.00%	2.01/20.37%	37.16/32.93%

‡ The left value corresponds to the 4iQ dataset, and the right value corresponds to the Naz.API dataset. Both results are obtained with one guess.

For the credential-based bucket mechanism, we perform joint guessing of usernames and passwords. This process incurs high computational overhead, similar to the cost of guessing salted passwords, and results in a lower success rate. Nevertheless, in Scenarios #1, #3, and #4, our attack successfully extracts between 20.37% and 32.93% of credentials on the Naz.API dataset, and between 1.62% and 37.16% on the 4iQ dataset, all with a single guess.

The impact of parameter w. Following observation 3 in Sec. 1, we introduce a parameter w in Algo. 1 to assign additional weight to repeated queries. Fig. 4 presents the results for $w = 0$ and $w = 0.5$ in scenario #1 under the password-based bucketization with $min\_count = 4$. As shown, both settings yield similar %Usage, but using $w = 0.5$ results in noticeably higher %Accuracy (see Table 4 for metric definitions). We select $min\_count = 4$ for the 4iQ dataset because more batch queries cannot reach four occurrences before pruning. The higher %Accuracy when $w = 0.5$ confirms that the parameter effectively boosts the counts of batch queries above the threshold, thereby validating the practicality of observation 3.

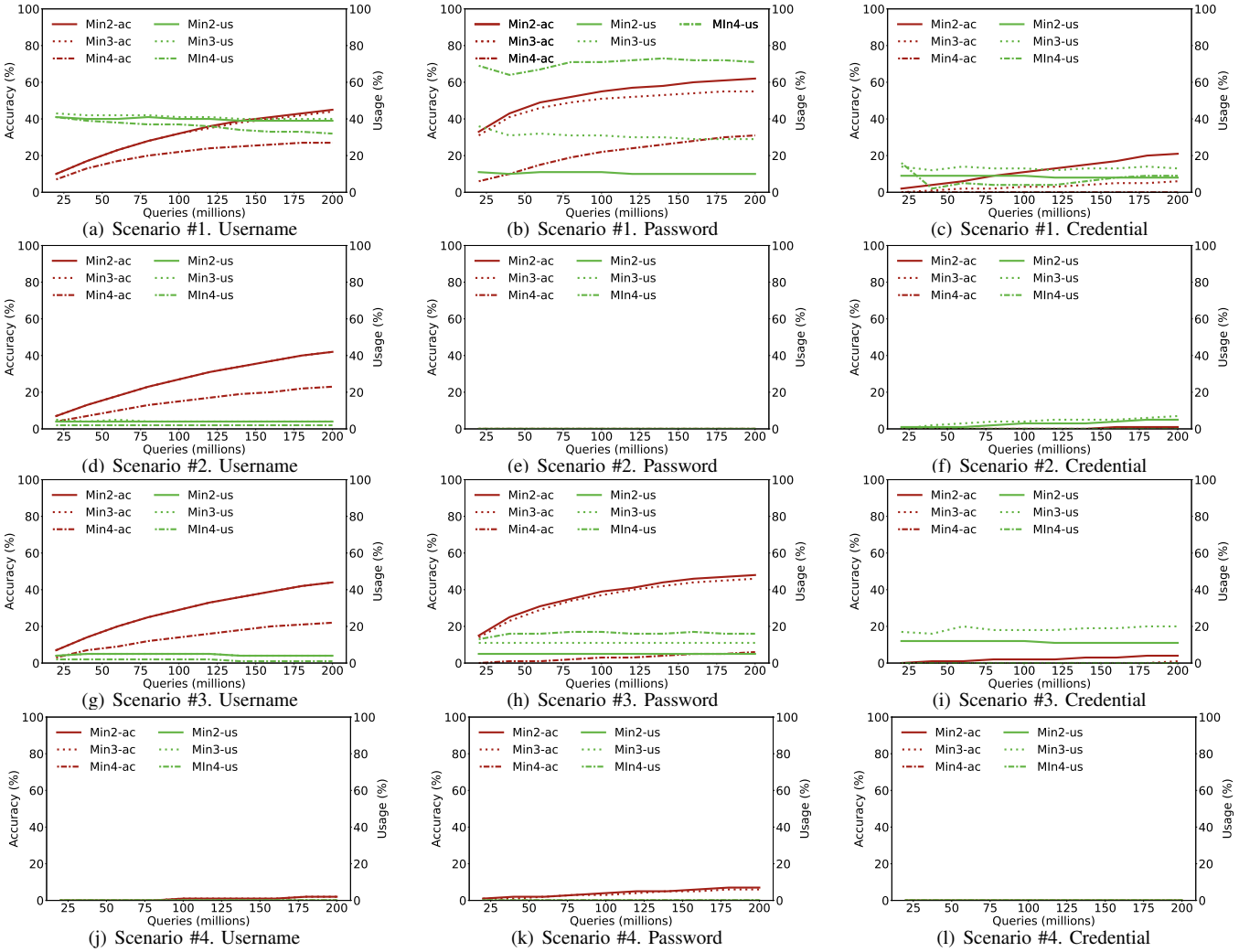
Detailed analysis of credential connecting. The results of credential connecting are presented in Table 8. The connected rate decreases as the $min\_overlap$ threshold increases. The highest connected rate, 89.31%, is observed in scenario #2 under the password-based bucket mechanism in the 4iQ dataset, while the highest in the Naz.API dataset is 27.73%, achieved in scenario #1, also under the password-based bucket mechanism. Across all scenarios, the username-based bucket mechanism exhibits the lowest connected rate. This is because the credential connecting algorithm cannot associate queries that fall within the same bucket. Moreover, users frequently reuse usernames, such as using the same email address across multiple websites, which results in identified queries being grouped into the same bucket, thereby limiting linkage across users.

The popular rate in the 4iQ dataset is significantly higher than that in the Naz.API dataset. This is because the Naz.API dataset contains real PM users, who tend to use fewer popular passwords compared to the synthetic users in the 4iQ dataset. Interestingly, in scenarios #2, #3, and #4 of the 4iQ dataset, both connected and success rate can exceed those observed in scenario #1. This occurs because scenarios #2 to #4 may include queries from ordinary users mixed into batch queries. In the 4iQ dataset, ordinary users are more likely to query popular passwords. For example, if a PM user's vault contains the leaked passwords `123456` and `Myleakedone1`, and the user later changes `123456`, his vault may still be connected if an ordinary user sends a query `123456` interspersed among a batch query.

The above reason also explains why the success rate in scenario #1 can be lower than in other scenarios. In scenarios #2 to #4, the number of PM users expected to be connected decreases due to vault changes. However, PM users can still be connected through popular passwords that appear within batch queries, which increases the success rate. The low success rate in scenario #2 under the credential-based bucket mechanism also reflects this phenomenon. Even when PM users have popular passwords, differences in usernames lead to different credentials, preventing successful connections.

Appendix D. Potential optimizations for our attacks

Potential optimizations for identification. According to the results in Fig. 2 and 3, our Algo. 1 demonstrates the



(a) Scenario #1. Username (b) Scenario #1. Password (c) Scenario #1. Credential

(d) Scenario #2. Username (e) Scenario #2. Password (f) Scenario #2. Credential

(g) Scenario #3. Username (h) Scenario #3. Password (i) Scenario #3. Credential

(j) Scenario #4. Username (k) Scenario #4. Password (l) Scenario #4. Credential

Figure 3: Experiments on subsequence identification with lengths ranging from 10 to 20 and min_count set to two, three, and four. The results report %Accuracy and %Usage across scenarios #1 to #4 under three different bucket mechanisms: username-based, password-based, and credential-based. The evaluation is conducted on the 4iQ dataset.

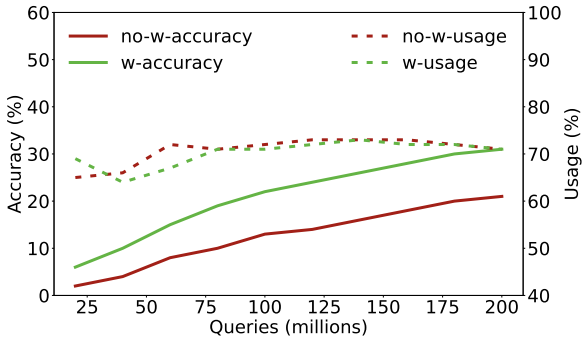


Figure 4: Comparison of %Accuracy and %Usage between $w = 0$ and $w = 0.5$ in scenario #1 under the password-based bucket mechanism on the 4iQ dataset ($min\_count = 4$).

practicality of the identification approach but also reveals opportunities for improvement. One potential enhancement is to incorporate similarity-based comparisons to identify additional queries. For example, consider a PM user's queries as (q_a, q_b, q_c, q_d). If the user updates one credential, the subsequent queries might be (q_a, q_b, q_c, q_e). These two query sets differ by only one element and exhibit a high degree of similarity. By employing similarity metrics, such as Jaccard similarity, we can improve %Accuracy. Furthermore, machine learning or deep learning techniques could be employed to capture contextual relationships between queries and their adjacent entries, potentially enabling more robust identification under noisy or dynamic query behaviors.

Potential optimizations for credential connecting. In our credential connecting algorithm, we rely solely on leaked information. To enhance the connected rate, similarity matching can be incorporated. Specifically, before matching credentials from the breach dataset to identified subsequences, the attacker could generate a series of similar credentials, such as those with high cosine similarity [42],

TABLE 8: Connected rate, popular rate, and success rate under varying $min\_overlap$ values across different scenarios.[†]

Scenario	Bucketization	Connected rate				Popular rate				Success rate			
		$op=2$	$op=3$	$op=4$	$op=5$	$op=2$	$op=3$	$op=4$	$op=5$	$op=2$	$op=3$	$op=4$	$op=5$
#1: 4iQ	Username	14.09%	12.38%	8.61%	3.52%	-	-	-	-	22.05%	22.45%	22.27%	22.76%
#1: 4iQ	Password	70.33%	45.46%	37.75%	36.44%	69.06%	69.81%	69.16%	69.64%	**94.69%**	**95.32%**	**96.05%**	**96.58%**
#1: 4iQ	Credential	32.51%	31.63%	29.95%	26.72%	84.00%	84.13%	84.31%	84.84%	8.58%	8.49%	8.49%	8.50%
#2: 4iQ	Username	5.38%	4.59%	3.71%	1.64%	-	-	-	-	64.73%	65.68%	64.16%	54.23%
#2: 4iQ	Password	89.31%	23.30%	11.16%	8.45%	62.10%	64.39%	63.32%	59.83%	**95.46%**	**92.79%**	**90.80%**	80.00%
#2: 4iQ	Credential	29.31%	15.11%	9.67%	6.95%	71.87%	78.41%	79.19%	78.83%	0.00%	0.00%	0.00%	0.00%
#3: 4iQ	Username	5.15%	4.39%	3.68%	1.82%	-	-	-	-	88.79%	89.26%	89.91%	87.46%
#3: 4iQ	Password	77.49%	39.15%	28.55%	26.73%	67.44%	69.66%	69.54%	68.63%	**94.83%**	**94.40%**	**95.06%**	**95.70%**
#3: 4iQ	Credential	86.72%	80.37%	69.21%	53.23%	76.52%	77.22%	77.52%	78.37%	3.13%	3.04%	2.71%	1.38%
#4: 4iQ	Username	2.80%	2.24%	1.61%	0.58%	-	-	-	-	**98.19%**	**98.47%**	**98.06%**	**92.65%**
#4: 4iQ	Password	63.41%	34.42%	26.33%	24.61%	68.17%	69.82%	69.81%	69.23%	**95.50%**	**95.09%**	**95.49%**	**96.17%**
#4: 4iQ	Credential	80.55%	63.86%	43.44%	28.85%	70.56%	71.90%	73.48%	74.41%	2.84%	2.49%	1.71%	1.27%
#1: Naz	Username	5.21%	0.63%	0.08%	0.01%	-	-	-	-	65.32%	65.44%	64.93%	66.13%
#1: Naz	Password	27.73%	20.71%	15.93%	11.71%	2.97%	2.61%	2.50%	2.45%	72.66%	69.37%	68.06%	67.38%
#1: Naz	Credential	24.92%	22.00%	17.97%	13.69%	2.65%	2.54%	2.53%	2.52%	62.97%	62.81%	62.83%	62.77%
#2: Naz	Username	1.32%	0.20%	0.03%	0.01%	-	-	-	-	**99.19%**	**98.29%**	**97.17%**	**93.22%**
#2: Naz	Password	18.97%	11.44%	6.87%	3.88%	4.58%	4.62%	4.81%	5.21%	16.46%	0.00%	-	-
#2: Naz	Credential	18.12%	12.21%	7.72%	4.59%	4.34%	4.16%	4.22%	4.52%	13.46%	0.00%	0.00%	-
#3: Naz	Username	1.34%	0.20%	0.03%	0.01%	-	-	-	-	**99.34%**	**98.79%**	**97.20%**	**93.44%**
#3: Naz	Password	12.03%	9.94%	7.93%	5.84%	3.78%	3.71%	3.75%	3.87%	**93.36%**	**91.64%**	89.50%	87.10%
#3: Naz	Credential	12.34%	11.16%	9.34%	7.15%	3.67%	3.68%	3.76%	3.86%	**90.50%**	89.33%	87.54%	85.26%
#4: Naz	Username	0.52%	0.08%	0.01%	0.01%	-	-	-	-	**99.98%**	**99.95%**	**99.20%**	**100.00%**
#4: Naz	Password	8.35%	6.24%	4.33%	2.64%	3.40%	3.27%	3.42%	3.57%	**97.69%**	**96.80%**	**95.29%**	**91.93%**
#4: Naz	Credential	12.19%	11.01%	9.05%	6.60%	3.30%	3.31%	3.33%	3.39%	70.09%	67.44%	64.10%	60.07%

[†] *op* denotes the parameter $min\_overlap$; "-" indicates that the denominator of the rate is zero; Bold values highlight results that exceed 90%.

for each user in the dataset. These generated credentials are then hashed and compared to the identified subsequences to improve the connected rate. As a result, the credential connecting may be able to associate users even when they have only one or no leaked credentials directly appearing in the breach dataset. However, this approach comes with increased computational and storage costs.

Appendix E. Password managers selection

We conducted a search using the terms "password manager", "password vault", "password management", and "password store" across the Microsoft Edge Add-ons Store, Firefox Add-ons Store, and Chrome Web Store. From each set of search results, we selected the top 30 PMs, removing duplicates across platforms. To ensure comprehensive coverage, we also included the built-in PMs of the three browsers: Edge, Firefox, and Chrome.

To refine the candidate pool, we applied two key criteria: (1) excluding PMs with fewer than 10,000 users to prioritize widely adopted solutions, and (2) excluding PMs that do not support C3 functionality. Beginning with an initial set of 57 PMs, this filtering process narrowed the selection to 14 PMs for detailed analysis, as summarized in Table 5.

Appendix F. Query patterns collection methodology

We collect information from PM documentation and conduct practical tests. Our analysis focuses on three key aspects: trigger conditions, query patterns, and potential information leakage (i.e., cookies). These aspects are essential for addressing our research questions.

We begin by navigating the official websites of the selected PMs to collect white papers and other relevant documentation. This allows us to determine which C3 services each PM integrates with and under what conditions compromised credential checks are triggered. We then register accounts for each PM, which are deleted upon completion of the study to minimize the load on PM servers.

For each account, we create a total of 90 credentials using fake usernames. Specifically, we generate 15 weak passwords (as classified by each PM), 27 reused passwords to simulate common user behavior, and the remaining passwords are randomly generated by each PM. These proportions are based on the distribution reported in [26].

We trigger automatic compromised credential checks for each PM and also perform basic operations, i.e., adding, changing, and deleting credentials, to explore whether these actions can trigger credential checking. During this process, we initiate packet capture using BurpSuite to collect all communication between the PMs and the C3 service.

After capturing the data packets, we perform a detailed analysis of the HTTPS traffic to identify the query patterns of each PM. We investigate whether PMs check all credentials each time and whether the sequence of queries remains consistent. Additionally, we examine whether stateful cookies are present in the data packets. This process is repeated every day for a total of seven times to detect any periodic patterns or behavioral changes over time.

Ethical considerations. Our methodology involves no exploitative actions, and all data collection occurs on the client side during normal usage of the PMs. All credentials in our registered accounts are simulated. After the information is gathered, we delete the accounts to prevent imposing unnecessary load on the PM servers due to inactive accounts.

Appendix G. Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

G.1. Summary

The paper presents attacks exploiting the interaction between Password Managers (PMs) and Compromised Credential Checking (C3) services. C3 services such as Google Password Checkup (GPC) and Have I Been Pwned (HIBP) allow users to make queries to determine whether or not particular password credentials are known to be compromised. A query to the C3 server includes a short (e.g., 20 bit) hash prefix (bucket) derived from the username and/or password. PMs utilize C3 services to check whether or not any of the passwords in a user's vault have been compromised. The paper finds that many PMs submit all of these C3 queries in a large batch and often fail to deduplicate C3 queries. The resulting query patterns leak information which enables an honest-but-curious C3 server to identify password manager users and link their credentials. This substantially improves attack accuracy compared to attacks that do not utilize this additional information channel. Empirical simulations demonstrate that query pattern leakage enable successful identification of 83.04% to 87.59% of password manager users. The attacks are most effective when the C3 server utilizes password based bucketing, but the attacks still have some impact when the underlying C3 protocol adopts username only based bucketing mechanism i.e., GPC.

G.2. Scientific Contributions

- Identifies an Impactful Vulnerability

G.3. Reasons for Acceptance

1) This paper identifies an impactful vulnerability which exploits the interaction between PMs and C3 services. The attack exploits actual behaviors of deployed password managers (13 out of 14 password managers send entire vaults in batch operations), making it practically relevant rather than purely theoretical. The paper describes simulations on datasets with 1.4 billion credentials and 300 million credentials and analyzed 14 popular password managers
2) Responsible Disclosure: The authors reported vulnerabilities to affected vendors, with 10 out of 14 PMs found vulnerable, and received responses from 7 companies.

G.4. Noteworthy Concerns

1) Reviewers noted that different bucketing mechanisms (e.g., password-based, username-based, or credential-based) can significantly affect the effectiveness of attacks. In particular, password-based bucketing mechanisms appear to yield the most effective attacks, while username-only bucketing schemes tend to be more resistant. Reviewers further emphasized that username-based bucketing protocols do provide strong protection for unbreached passwords, provided that those passwords are chosen independently of any previously breached credentials.
2) Artificial restrictions on user populations (10-20 credentials) may not represent real-world scenarios.
3) The attack effectiveness varies significantly between datasets (4iQ vs. Naz.API) due to different password popularity distributions.