Safeguarding Online Authentication Systems from Attacks

by

Mazharul Islam

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2025

Date of final oral examination: 02/21/2025

The dissertation is approved by the following members of the Final Oral Committee:

Rahul Chatterjee, Assistant Professor, Computer Science
Michael Swift, Professor, Computer Science
Kassem Fawaz, Associate Professor, Electrical and Computer Engineering
Rishab Goyal, Assistant Professor, Computer Science

To love of my life
My wife Salsabil
My son Ammar

ACKNOWLEDGMENTS

"They planned, but Allah also planned. And Allah is the best of planners."

Sura Al-Anfal, Verse 30

First and foremost, I would like to express my deepest gratitude to the Almighty Allah for guiding me on the right path and granting me the strength and opportunity to complete my Ph.D. At the start of my Ph.D., I did not know that Allah had rewritten such a rewarding journey for me.

This dissertation would not have been possible without the support of my Ph.D. advisor, Prof. Rahul Chatterjee. Rahul is a great human being and has helped me immensely, often going out of his way to support me in both research and personal growth. Rahul, thank you for your patience, mentorship, and unwavering support throughout my Ph.D. journey.

I am also grateful to my labmates at UW–Madison——Majed Almansoori, Paul Chung, Sophie Stephenson, Rose Ceccio, Naman Gupta, Michelle J., Tim Lu, Julia Nonnenkamp, and many others. Being surrounded by such lively, spirited, and colorful individuals has made my Ph.D. journey an enjoyable and enriching experience.

I extend my sincere gratitude to Prof. Thomas Ristenpart from Cornell Tech for his continuous support and invaluable guidance. Whenever I encountered roadblocks in my research, he provided insightful ideas and rich in-deepth, thinking that has helped me navigate though many technical challenges. Tom, thank you for being an incredible mentor and for shaping the way I think as a researcher.

During my Ph.D., I had the privilege of collaborating with two of Tom's Ph.D. students, Dr. Marina Sanusi Bohuk and Dr. Bijeeta Pal. I am especially grateful to Marina for her immense contributions in de-

signing and shaping the first three chapters of this dissertation. Marina has been a fantastic collaborator, and our partnership has spanned over four years—while I focused on implementation, design, and analysis at UW–Madison, she handled deployment and analysis at Cornell University. Parts of this dissertation are a direct result of our joint efforts, and I deeply appreciate her expertise and dedication. Similarly, I am thankful to Bijeeta, who was the lead author of my first Ph.D. project, for her relentless support and contributions.

I also had the opportunity to intern at Visa Research for three summers, and I am deeply grateful to the people there, especially my colleagues and friends from the Identity and AI Security team — Dr. Ke (Coby) Wang, Dr, Sunpreet S. Arora, Dr. Maliheh Shirvanian, and Dr. Shimaa Ahmed. Coby, in particular, has been instrumental in guiding me throughout my Ph.D. journey, and our collaborative work forms the foundation of the final chapter of this thesis. Coby, whether I needed advice, friendship, or support, you were always there for me——thank you.

I am also grateful to my Ph.D. committee members, Prof. Michael Swift, Prof. Kassem Fawaz, and Prof. Rishab Goyal, for their invaluable feedback and guidance, which have helped improved this dissertation.

Lastly, and most importantly, I want to express my deepest appreciation to my beloved wife, Salsabil Arabi. Words will always fall short in capturing how much you mean to me and the immense sacrifices you have made throughout this journey. Just know that I love you and I am greatful to have you in my life.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Compromising user accounts is one of the easiest and most common ways for attackers to undermine otherwise secure online services, often leading to severe harm. According to Verizon's Data Breach Investigations, 89% of web application hacking attempts involve credential abuse. The consequences can be catastrophic—such as the Colonial Pipeline attack, which caused $3 million in damages due to a single leaked password. To safeguard user authentication from such damaging attacks, in this thesis I propose effective detection mechanisms that leverage safely and securely collected characteristics of user-submitted credentials.

I first present Gossamer, a framework that collects a rich set of characteristics from user-submitted passwords in a safe and secure manner. Then, I propose Araña, a filter- and clustering-based attack discovery pipeline framework that can uncover stealthy, hard-to-spot attack campaigns and characterize attacker behavior. Together, these two frameworks enabled us to collect 34 million login requests over seven months at two large universities, leading to the discovery of 29 attack campaigns and 1,183 compromised accounts. I further highlight how Araña can be customized for timely attack detection and detail a new data-driven approach to measure the extent to which the proposed detection mechanism can withstand attackers with additional resources.

Moving on to passkey-based user authentication, I propose a new detection framework called CASPER, which allows online services to detect whether attackers are using leaked passkeys stolen from cloud backup storage to compromise user accounts. We present a decoy-based cryptographic detection protocol that enables online services to distinguish real passkeys from decoy passkeys. I detail how CASPER can be seamlessly integrated into real-world systems with minimal impact on user experience, negligible performance overhead, and minimal deployment

and storage complexity for participating parties. Finally, I evaluate the detection accuracy of CASPER against optimal attackers who attempt to bypass detection by leveraging data breaches from other websites.

1 INTRODUCTION

Online services rely on user authentication systems to protect user accounts and safeguard the services they provide from unauthorized access. However, attackers are highly motivated to target these authentication mechanisms, as they often serve as the most accessible entry point for compromising otherwise secure services. Recent reports from Verizon's Data Breach Investigations highlight that 89% of web application hacking attempts involve credential abuse [1]. These attacks, if successful, can cause catastrophic damage and irreparable harm.

Password-based authentication remains the most widely used and user-friendly method for online authentication. However, many online services, including Apple, Google, and Microsoft, are transitioning toward passkey-based authentication [2]. Passkeys are cryptographic authentication credentials generated by users' devices, and allow users to log onto their account without requiring them to type any passwords. In both authentication systems, during registration, the user or their device generates a secret credential, while the authentication server stores corresponding verification data in a credential database. An attacker without knowledge of the secret credential should not be able to bypass the authentication server's verification checks. Unfortunately, both authentication systems are susceptible to attacks due to the large-scale leakage of credential database.

Password leaks often result from breaches of credential databases storing salted hash of user chosen passwords from authentication servers. Since 2013, the service "Have I Been Pwned" (HIBP) [3] has collected over 5.5 billion leaked passwords. Over the last few decades, breaches from authentication server have become increasingly common for passwords, The vast number of exposed passwords, combined with users' tendencies to reuse or select weak passwords, has enabled attackers to launch highly effective online guessing attacks. A marked example of this was the Colo-

nial Pipeline attack [4], which resulted in $3 million in damages due to a single leaked password. Existing mechanisms to detect malicious logins due to such leakage of users passwords remains ineffective today, and fails to protect users account from unauthorized access.

Unlike passwords, passkey-based authentication does not expose users' passkeys even if the credential database from an authentication server is breached. This is because the verification data stored by authentication servers should not reveal any information about the actual passkeys of the user. However, many credential backup services such as iCloud Keychain from Apple [5], Google Password Manager [6], Password Monitor from Microsoft [7], 1Password [8], LastPass [9], and DashLane [10] keep a cloud backup storage of users passkeys to allow users to recover them if they lose access to their devices. Passkey security is threatened when such cloud backup storage of users passkeys gets compromised. As a result attackers can steal users' passkeys by breaching credential backup services, potentially compromising accounts across multiple services. This can occur if an attacker gains access to a user's backup account (e.g., by guessing its password) or if an insider attacker accesses the cloud storage. Unfortunately, existing credential backup services lack the capability to detect unauthorized access to users' cloud backed-up passkeys.

In this thesis, I aim to develop detection mechanisms that enable authentication servers to reliably identify malicious login attempts in online authentication systems. The core hypothesis driving this research is that *rich characteristics of submitted credentials can be collected safely and securely to improve malicious login detection*.

To build such improved detection system, several critical questions must be answered that are not well-addressed in existing literature:

(1) What characteristics of submitted credentials should authentication servers log to provide strong and accurate signals for detecting malicious logins?

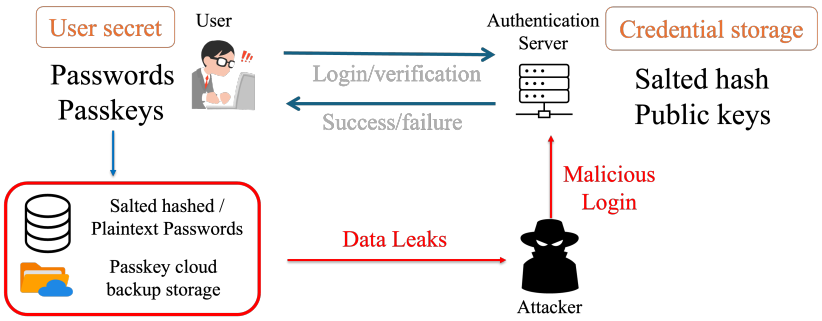


Figure 1.1: Password leaks from authentication servers and passkey leaks from credential cloud backup services enables attackers to send malicious login attempts to take over users' accounts. We aim to develop effective malicious login detection approaches that allows authentication servers to detect majority of the malicious logins detect while minimizing false alterts.

(2) How can these characteristics be collected in a safe and secure manner without compromising user account security?

(3) How can we develop an effective malicious login detection system that minimizes false positives while capturing the majority of attacks?

(4) How robust will such detection mechanisms be against motivated attackers with more knowledge and resources?

This thesis systematically answers these questions for both password and passkey-based authentication.

In Chapter 3, we introduce Gossamer [11], a system that enables authentication servers to log a carefully chosen set of characteristics from user-submitted passwords. Through a simulation-based analysis, we demonstrate that these logs do not endanger users' account security, even if compromised.

We continue our work on identifing malicious logins for password based user authentication in Chapter 4 by presenting Araña — an attack

campaign discovery framework. Araña is a filter-cluster-based pipeline that allows authentication servers to detect and identify malicious login attempts exhibiting similar behavioral patterns. We show how Araña helped us in identifing large scale hard to detect attack campaigns by leveraging Gossamer collected characteristics from user submitted passwords. We conclude the chapter by discussing real world constraints that may arise during deployment of Araña and discuss briefly about a data-driven approach for evaluating the robustness of our malicious login detection methods.

In Chapter 5, we focus on passkey-based authentication and due to its fundamental differences from password-based authentication we employ a decoy based detection technique. Specifically to enable the websites detect unauthorized login attempts using passkeys stolen from breached credential cloud backup services, we propose a decoy-based detection technique. Our method involves adding a new characterisitc to the passkey before they are uploaded to a passkey cloud storage — hiding real passkeys within a list of decoy passkeys that are indistinguishable from genuine ones. As a result when attacker compromise users cloud passkey storage, and use the stolen passkeys to compromise users' account the websites / online services will be able to detect such maliicous logins. This technique provides a strong and accurate signal to authentication servers about potentially malicious login attempts. Additionally, we evaluate the robustness of CASPER against attackers who have breached other websites and strategically attempt to avoid detection.

In summary, this thesis presents novel techniques for detecting malicious login attempts in both password-based and passkey-based authentication systems. By systematically designing and evaluating detection mechanisms, we aim to enhance the security of online authentication systems against credential abuse. We conclude this dissertation by discussing promising open research directions in the authentication space in

chapter 6.

1.1 Contributions

We describe the contributions of each chapter of this dissertation as following.

Securely Measuring Password-based Logins

In Chapter 3, we propose a system called Gossamer [11] that enables online services to securely log characteristics about submitted login attempts. Via a simulation based approach we carefully chose statistics that will not endanger users' account security should the logged characteristics of submitted passwords gets leaked and yet will be valuable in identifying malicious logins. We work in collaboration with IT-offices of two US universities and deployed Gossamer for seven months collecting 34 million login requests.

Discovering and Characterizing Password Guessing Attacks in Practice

In chapter 4, we perform the first in-depth analysis of a dataset including over 34 Million login events generated by Gossamer in order to discover and characterisitc remote password guessing attack. we present an analysis framework called Araña [12] that illustrates how to filter and cluster Gossamer logs to enable easy manual analysis and identify attack campaigns. We use Araña to identify 29 attack clusters against two US universities where Gossamer was deployed. We identify key characteristics and patterns of attacks received by authentication systems at these universities, and we discuss how authentication systems should evolve to counter such threat. We finish this chapter by discussing real world considerations that

may arise durign the deplopying Araña such as how to customize Araña to detect attack in a timely manner and how to evaluate the robustness of the proposed approach.

Detecting Compromise of Passkey Storage on the Cloud

In Chapter 5, we propose CASPER, the first framework to detect the abuse of FIDO2 synced passkeys leaked from passkey management service (PMS) providers. Importantly, CASPER can also be easily extended to detect breaches of other cryptographic credentials that are widely used today, such as HMAC-based / time-based one-time passwords (HOTP / TOTP) seeds. We show how CASPER enables the websites to detect malicious logins effectively using passkeys stolen from PMS providers. Furthermore we analyze its detection effectiveness systemically against sophisticated attackers who tries to evade detection by leveraging breaches from other already breached websites

In this chapter, we describe CASPER's carefully design protocol that can be seamlessly integrated into the existing passkey backup, synchronization, and authentication processes, with only minimal impact on user experience, negligible performance overhead, and minimum deployment and storage complexity for the participating parties. Finally via a prototype implementation of CASPER, we confirm that CASPER introduces negligible performance and storage overhead for all parties involved and demonstrate that deploying CASPER requires minimal modifications to PMS and RPs.

2 BACKGROUND

In this chapter, we present background topics relevant to the work presented in this dissertation. Specifically we start by discussing the current state of passwords and the motivation for passkeys. We then introduce the concept of synced passkeys and the tension between security and recoverability. Finally, we discuss the directed anomaly scoring (DAS) approach used in this dissertation to identify malicious IP addresses.

2.1 Passwords

Current login systems still heavily rely on password-based authentication. Users typically enter their usernames and passwords to a form on a web client, which submits them along with other information relating to the user or machine such as HTTP headers, cookies, IP, and user agent to the login server over HTTPS. The server hashes the password (and a salt), checks if the username and hash pair is present in the login database, and if so, allows the user to log in or prompts for further authentication checks. Otherwise the request fails.

Single sign-on (SSO) systems allow a user to log into multiple different web services using the same username and password. When a user accesses a service, the service provider (SP) redirects the user to obtain a proof of authentication from the identity provider (IdP). The IdP provides the proof immediately if the user has recently authenticated with it, or requires the user to authenticate and provides the proof if the authentication is successful. The OAuth framework [13] is a common way to achieve SSO.

2.2 Passkeys

FIDO2 authentication method

FIDO2 consists of a set of sub-protocols (e.g., CTAP 2.0 [14] and WebAuthn [15]) with digital signature schemes serving as its cryptographic heart. Briefly, during registration, the user's authenticator device creates a signature key pair (s, v), where s and v are the private signing key and public verification key, respectively. Only v is sent to the website while s stays private on the authenticator. During the login phase, RP sends a challenge to the authenticator, and the authenticator then uses s to produce a response to the challenge. Finally, the RP can use the corresponding verification key v to verify the response and decide whether to grant the user account access.

FIDO2 authentication is not only resistant to guessing and phishing attacks, which traditional passwords have been long suffering from, but also protects users' accounts against RP data breaches — a breached v reveals only a negligible amount of information about its corresponding s if the underlying digital signature schemes are secure, and importantly, nothing about users' credentials for other RP accounts.

Account recovery concerns in FIDO2

The improved security guarantees of FIDO2 [16, 17, 18] have prompted their adoption either as a single-factor replacement for passwords [2] or itself as multi-factor authentication (MFA) [19]. However, the lack of secure and user-friendly account recovery solutions for the key management required by such cryptographic protocols has been a source of serious user frustration and concerns [20, 21, 22, 23, 24, 25, 26, 27]. This happens when users' FIDO authenticators — the devices to which users' FIDO2 private signing keys are bound — become permanently unavailable due to device

loss, reset, or theft. For example, recent studies [27, 23] identify the lack of convenient and secure account recovery options for FIDO2 passwordless authentication as a major obstacle to eliminating passwords on the web in the foreseeable future. A study by Las et al. [22] found that over 60% of the participants expressed serious concerns about secure account recovery in enterprise settings. Some RPs attempt to resolve such user frustration and concerns by offering easy-to-use yet insecure account recovery or backup authentication options, e.g., user-chosen passwords, email/phone-based one-time codes, or secret questions, for FIDO authentication. This, however, overshadows the security benefits provided by FIDO authentication and degrades the overall security of users' accounts [28]. To tackle the account recovery problem, industry-led efforts have emerged to encourage the adoption of synced passkeys, which we will introduce next.

Synced passkeys

Synced passkeys are rebranded FIDO2 private signing keys, with passkey management services (PMS) synchronizing them between PMS's cloud storage and users' authenticators. For conciseness, in this thesis, we use "passkey" to refer specifically to a "synced passkey" as opposed to a "device-bound passkey" — a FIDO private key that never leaves the authenticator. . Examples of PMS Today include, Apple's iCloud Keychain [5], Google Password Manager [6], Microsoft's Password Monitor [7], 1Password [8], LastPass [9], and DashLane [10]. These services are supported by all major browsers, and used by millions of users [29].

The tension between security vs. recoverability

Due to their centralized nature in storing users' private signing keys, passkeys have inadvertently become a lucrative target for attackers. Many PMS now use another cryptographic key to protect a user's passkeys

backup at rest, for example, by encrypting it with another key. This encryption key needs to be consistently available for users; otherwise, users will not be able to decrypt the encrypted passkeys after retrieving them from passkey providers. In other words, if not properly addressed, managing such a cryptographic key, which has long been a daunting task for users [30], inevitably reintroduces the account recovery problem that synced passkeys attempt to solve in the first place.

For this reason, some PMS only require users to maintain a user-chosen secret (e.g., passwords or PINs in most cases), which is presumably more memorable, recallable, and hence recoverable than a high-entropy cryptographic key. In practice, this secret can be used in two flavors of strategies, i.e., key derivation based and key escrow based, with each emphasizing different priorities on the security and recoverability of the encryption key. However, both strategies rely on user-chosen secrets that are subject to guessing attacks (e.g., offline cracking), making them insufficient for protecting passkeys. If PMS is compromised and the user-chosen secrets used to derive or to retrieve the decryption key are guessed, attackers can access users' passkeys in plaintext.

2.3 Directed Anomaly Scoring (DAS)

In Chapter 4 of this dissertation, we use directed anomaly scoring (DAS) introduced by Ho et al. [31]. Conceptually DAS is a technique that compares two sets of events and flags the one which one is more suspicious than the other. In a bit more detail, DAS approach involves first choosing a set of features that describes the suspiciousness of an event and labeling for each feature which direction — higher or lower — is "suspicious." For example, the higher the number of requests from an IP address, the more likely it's malicious.

We leverage this DAS scoring to identify malicious $\mathcal{L}$ sets as defined

in Chapter 4. Specifically to score $\mathcal{L}$ sets against each other, we use DAS scoring. For example number of failed login requests (NR) can be a DAS feature with higher being the direction of maliciousness. Then DAS uses these per-feature comparisons to define an $\mathcal{L}$ set comparison: $\mathcal{L}$ set A is more suspicious than $\mathcal{L}$ set B if all of A's feature values are more malicious than the respective feature values of B. Finally, the *DAS score* of an $\mathcal{L}$ set compared to a set of other events is equal to the number of other events for which it is more suspicious.

3 SECURELY MEASURING PASSWORD-BASED LOGINS

Summary: Passwords remain the primary way to authenticate users online. Yet little is known about the characteristics of login requests submitted to login systems due to the sensitivity of monitoring submitted passwords. This means we don't have answers to basic questions, such as how often users submit a password similar to their actual password, whether users often resubmit the same incorrect password, how many users utilize passwords known to be in a public breach, and more. Whether we can build and deploy measurement infrastructure to safely answer such questions is, itself, an open question.

We offer a system, called Gossamer, that enables securely logging information about login attempts, including carefully chosen statistics about submitted passwords. We provide a simulation-based approach for tuning the security-utility trade-offs for storing different password-derived statistics. This enables us to gather useful measurements while reducing risk even in the unlikely case of complete compromise of the measurement system. We worked closely with two large universities and deployed Gossamer to perform a measurement study that observed 34 million login requests over a seven month period. The measurements we gather provide insight into the use of breached credentials, password usability, and other characteristics of the submitted login requests.

Acknowledgements

This chapter is based on the paper published at USENIX Security conference 2022 [11]. I would like to express my sincere gratitude to all authors of the paper especially Dr. Marina Sanusi Bohuk for her contrbutions. Specifically while I handled deplyoing Gossamer and analyzing its the collected logs at UW–Madison, Dr. Bohuk led the efforts in in designing

Gossamer, and deplopying, analyzing its data collected at Cornell Tech.

3.1 Introduction

Despite the prevalence of password-based authentication across the internet, little is known about the passwords submitted to login systems. Knowing the characteristics of such login information would help practitioners make better security policies to improve both usability as well as attack detection. A key challenge hindering progress is that passwords are highly sensitive, and as a result prior work has only performed very limited measurements.

Two prior works are particularly relevant. Bonneau et al. [32] instrumented Yahoo login servers for 48 hours to learn the distribution of actual user passwords. But his technique could not record other information about submitted (valid or invalid) passwords, Chatterjee et al. [33] were the first to investigate incorrect password submissions from the viewpoint of a login server. They instrumented Dropbox's login service for 24 hours to investigate how often users submit a fixed set of easy-to-correct typos. However, their study was limited to only a specific set of typos, and does not provide a general framework for analyzing submitted passwords. Thus the question remains: Can we build login measurement infrastructure that monitors password submissions, but doesn't endanger security?

In this work, we design, build, and deploy a measurement system, called Gossamer, that securely records login requests, including statistics about submitted passwords. Doing this safely required extreme care, and our main contribution is a holistic approach that combines systems security features, a simulation-based framework to guide selection of password-derived statistics, and procedural safeguards. Ultimately, our initial deployment at two large universities is able to answer, for the first time, basic questions about submitted passwords—such as how often

legitimate users are making typos or repeatedly submitting the same incorrect password, whether attacks are detectable as credential stuffing, and more.

Performing such measurements requires jointly analyzing passwords submitted at different points in time. Prior measurement studies computed a (keyed) hash over a correctly submitted password [32] or compared the hashes of a small handful of variants of a submitted password to the real user's password hash [33]. Neither approach allows inferring whether users are submitting the same password multiple times or, if not, how many unique passwords they submit.

To enable such measurements, Gossamer's design uses a two-service logging infrastructure to ensure least privilege. Gossamer has a specialized *measurement service* that receives a copy of login requests from login servers, processes them by computing password statistics and encrypting submitted usernames, and outputs sanitized logs to a persistent database on a different machine. The measurement service, like login servers, has access to plaintext passwords. Thus we designed it to match or exceed the security properties of login servers: It is safe-on-reboot [34] (no sensitive data such as passwords are ever stored on disk), deletes all in-memory data periodically to limit the scope of what would be exposed in the case of a breach, and is administered by the same security staff in charge of login servers.

Researchers use a separate *analysis service* to access the sanitized logs stored in the persistent database. The sanitized logs and analysis service are still treated as sensitive, and cannot be made publically available. To assess the risk to user passwords in the unlikely case of complete exposure of both a login system's password hash databases plus Gossamer logs, we developed a new simulation-based approach to analyze the speed-up of brute-force cracking attacks that attempt to additionally exploit Gossamer logs. For example, simulations show that storing raw strength scores (as

measured by zxcvbn [35]) can provide up to a 20% increase in cracking efficacy (using up to 10^9 hash computations), leading us to reduce the granularity of strength scores. Ultimately, our simulations suggest that the best performing attack increases password recovery rates using Gossamer logs by less than 2% using 10^9 queries.

To showcase the utility of Gossamer, we worked in close collaboration with two large universities' information technology (IT) security departments to perform a measurement study of login behavior. Our measurement study protocols, including Gossamer's design and implementation, went through a thorough, multi-step review process that included reviews by the security engineering teams from both universities, representatives of each university's administration, and the relevant IRBs. This process culminated in a determination that Gossamer poses minimal risk. We deployed Gossamer for seven months at University 1 (U1) and for three months at University 2 (U2). We observed 34 million login requests (combined) for approximately 500 K users who regularly log in to access various university-provided critical online services such as email, course enrollment, and employment information.

This enables first-of-their-kind measurements of password usability and security. We saw that 1.9% of valid users at U1 and 4.6% at U2 changed their password in the data collection period. We found that 6.5% of usernames at U1 appearing in public breaches are still using a password that is only a small variant of one of their leaked passwords. This motivates deployment of password breach alerting services that take into account similarity [36]. On the usability front, while the Dropbox study reported that 5% of failed attempts were due to easily correctable typos, our measurements indicate that 65% of failed attempts could be typos (within edit distance two from the actual password), suggesting this is a much larger cause of user frustration than previously imagined. We also report on the rate of login retries, the success and failure rate of app-based two-factor

authentication, and the possible adoption of password managers. Finally, we are able to report a few high-volume attacks, with insights enabled by Gossamer to characterize the attacker behavior involved.

Summary. In summary, this chapter proposes a measurement framework for passwords that can safely help answer basic questions about password use. Our contributions include:

- Design of Gossamer, which combines systems security, simulation-based selection of password statistics, and procedural safeguards to enable measurement studies of password-based login behavior.
- We worked with two large universities' IT departments to deploy Gossamer for multi-month measurement periods.
- We report for the first time on a variety of aspects of password-related usability and security, and discuss the implications of these measurements. For example, our measurements motivate the need for password breach alerting, suggest ways to improve lockout mechanisms, and more.

Finally, we hope that Gossamer can serve as a platform to help drive future research on improving usability and security of passwords. As such we are releasing Gossamer as a public, open source project that may be useful for security researchers both in industry and academia.

3.2 Related Work

In this chapeter we perform measurements at two large universities. Both U1 and U2 use SSO with Microsoft Active Directory Federation Service (ADFS). While at U2 all login traffic goes through ADFS, at U1, only a portion of traffic is via ADFS. This is part of the reason we see a lower rate of logins per day at U1 compared to U2 (Section 3.4).

Studies about passwords. Prior works [37, 38, 39, 40, 41] have investigated guessability of user-chosen passwords. Most of these rely on

breached password data to understand the distribution of user-chosen passwords. Several studies have used Amazon Mechanical Turk (AMT) to understand user choice of passwords [42, 43, 44], under different factors such as password requirements [42], presence of a password strength meter [43], and use of a password blocklist [45].

As passwords chosen in this environment may not represent passwords on real websites, several studies have inspected real user passwords through client-side, server-side, or offline instrumentation [46, 47, 48]. On the client side, Florencio and Herley [46] installed a Windows Live Toolbar component for five hundred thousand volunteer participants and analyzed their password behavior over 85 days. Similarly, Forget et al. [47] created a client-side data collection tool to observe user's password behavior in its natural environment [47, 48].

Measurement studies with login systems. To our knowledge, three studies have looked at user passwords by instrumenting the login servers. Bonneau et al. [32] instrumented Yahoo's login servers to receive login requests (including user passwords) and construct histograms of password characteristics based on user demographics. Mazurek et al. [49] correlated password strength with demographic information in an offline study with reversibly encrypted passwords on an access-restricted computer. Chatterjee et al. [33] instrumented the login code at Dropbox for failed login attempts to test whether applying a typo correction to the submitted password would have produced the correct password.

Open questions. Many open questions remain about the characteristics of the passwords submitted to a login system. For example, how often do users log in from multiple devices, how often do users submit the same incorrect password multiple times, and how often do users submit passwords that are similar to one of their leaked passwords? More importantly, can we collect information about the submitted password that allows analyzing login characteristics without degrading the security

of user passwords? Such a framework would help practitioners make data-driven login security policies, such as account lockout thresholds, that better balance between usability and security.

3.3 Designing a Secure Measurement Architecture

To analyze the passwords submitted to a login system, we need to instrument live login services and monitor login requests, including the submitted username and passwords. User passwords are highly sensitive and should never be logged. We therefore designed a secure instrumentation architecture that preserves the privacy of login requests while allowing meaningful analysis. We refer to it as Gossamer and deploy it at two login systems used at two universities in the United States. Gossamer is designed in close collaboration with the security engineers at these universities. Below we describe the built-in security considerations in our design and the integration with the existing login infrastructure at these universities.

The architecture. Gossamer enables instrumentation of typical web login servers, such as those used for single sign-on (SSO). An overview of Gossamer's architecture appears in Figure 3.1. A lightweight hook is deployed within the login server that, on every received login request, sends a stripped-down copy of the request to our instrumentation infrastructure on a separate, in-network machine. This is done using a separate thread to avoid any noticeable latency impact by the instrumentation on login behavior. A login request includes the username, password, IP address, a subset[1] from the HTTP header, as some content can be more sensitive than user passwords. For example, an "authentication cookie" could bypass MFA requirements. of the HTTP headers, timestamp, login result (success

[1]The login server removes sensitive cookies specified by the security engineers

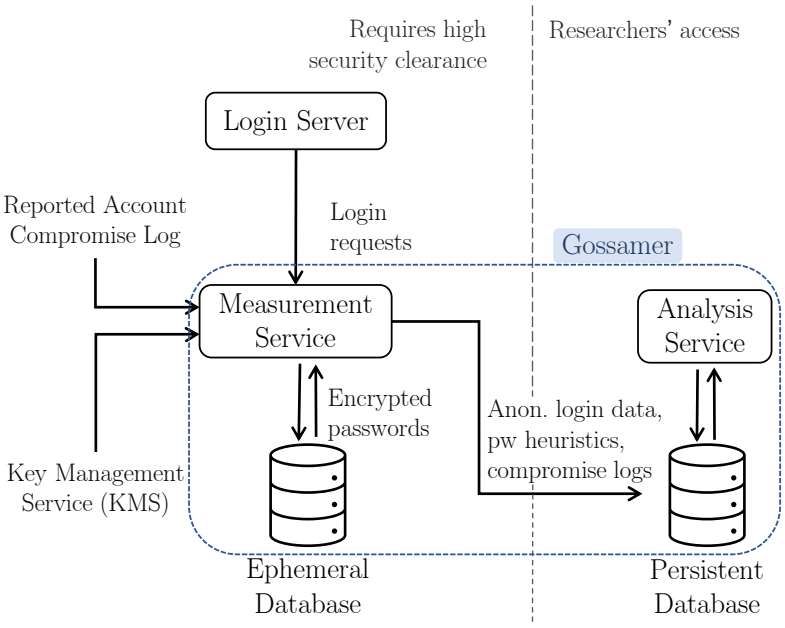


Figure 3.1: The main components of Gossamer.

or failure), and finally an application-specific result code for the login attempt.

A *measurement service* receives this forwarded login information. It is responsible for processing the raw login data in a secure manner, converting it into sanitized logs, and storing them in a persistent database. The persistent database can be accessed by analysts (in our case, researchers) via a dedicated *analysis service* for understanding user login behaviors. As such, we partition Gossamer's architecture into two security levels: The lightweight login hook and measurement service run at a higher privilege level and are administered by IT security staff; the analysis service is instead at a lower privilege level, accessible by analysts (researchers).

We explain more about these two services further below, but first describe our security and design goals.

Security properties and design goals. We design Gossamer to resist a variety of attacks. We note that all our network traffic is encrypted using

TLS, and as such we do not discuss network adversaries further. Instead, we focus on the threat of complete compromise of each (or both) of the services, as well as the weaker adversarial threat of exposure of logs generated by Gossamer.

To protect against these threats, we design Gossamer to conform to four security properties.

- *Least privilege access to password data.* The system must ensure that the analyst receives only the information necessary for analyzing login behaviors, while plaintext passwords remain restricted to particular services.
- *Bounded-leakage logging.* The system persistently logs a small set of statistics about user passwords. The set of statistics is carefully designed to bound the improvement in guessing attacks against user passwords, even in the case of complete compromise of the analysis service.
- *Periodic deletion.* The system should expunge all raw, sensitive data older than 24 hours to reduce the exposure of any data should the system get compromised.
- *Safe-on-reboot*. Finally, we must ensure that all sensitive data from raw HTTP requests is destroyed on reboot. This property was first introduced in [34] for the *Bunker* secure network tracing system.

We will return to these properties as we elaborate on the details of the architecture.

Security considerations in Gossamer. As mentioned, Gossamer uses two services running at different privilege levels on two different machines — a measurement service for processing the raw login data and storing the anonymized statistics of user logins into a secure, persistent relational database, and an analysis service for analyzing the data from the persistent database. Separating measurement from analysis enables us to maintain the same privilege requirements for access to password data as there

are without Gossamer. The high-level architecture diagram of Gossamer including privilege levels is shown in Figure 3.1.

The measurement service runs on a heavily access-restricted machine that receives a copy of the login request over an encrypted channel from the login servers. The service then computes measurements over the submitted password and stores them in the *persistent database*. Some interesting statistics require the plaintext submitted password across multiple requests — for example, the number of unique passwords submitted by a single user or from a single IP address. Therefore, the measurement service stores passwords encrypted using an in-memory key in the *ephemeral database*. The key is stored only in memory and is automatically replaced with a new key every day at midnight local time. The ephemeral database is placed in a memory-based file system, such as the `/tmp` directory. The key rotation cryptographically erases the data stored in the ephemeral database every 24 hours.[2] If the measurement service is killed or the device is rebooted, all ephemeral data is effectively deleted. This ensures our periodic deletion and safe-on-reboot requirements.

The ephemeral database allows us to calculate a number of measurements referencing the passwords submitted across multiple logins. These measurements (given in Section 3.4) allow us to characterize user behavior and could help in building attack detection mechanisms. The output of the measurement is stored in a persistent database outside the privilege boundary, where it can be accessed by the researchers. This database is placed in a disk-encrypted volume, providing another layer of protection in case the volume is backed up to an unprotected machine or is compromised. The key to the encrypted volume is only known to a subset of

[2]Key rotation at midnight deletes the data received, say, an hour before midnight, limiting our ability to correlate between passwords received before and after midnight. It is an open question how to design an efficient key-rotation technique that will allow secure deletion without requiring storing linear number keys in memory.

researchers, as the IT security engineers did not need it.

Protecting user privacy. To protect the privacy of users, Gossamer anonymizes the usernames before storing in the persistent database by encrypting them using a deterministic encryption scheme [50]. The encryption key is only accessible from within the measurement service. The deterministic nature of encryption allows us to cross-reference the logins against a username without knowing the actual username, while also allowing us to report compromised usernames to the security engineers, should we discover any. We do not record any personally identifiable information about the user, including their real name, affiliation, or account type (such as student, faculty, or staff). We do record the source IP address for requests, which is needed to analyze client and attack behaviors.

Of course re-identification attacks [51, 52] may be possible given access to these logs, and for this reason alone logs are not suitable for public release. We have strict policies against re-identification for the limited set of researchers who access the analysis service. All analysis is performed on the analysis server with encrypted usernames, and only summary statistics leave the analysis service.

Both the persistent and ephemeral databases are instantiated as MySQL databases. The measurement service is a Python Flask application running on an Apache server. We use the Python Fernet [53] library to encrypt user passwords in the ephemeral database using AES-256, and we use the Python Miscreant [54] library to deterministically encrypt and decrypt the usernames using AES-SIV.

Integration with other data sources. Looking ahead, in both of our deployments there are other relevant data sources available that we would like to analyze. Namely, it is common for organizations to have a database that contains reports about potentially compromised accounts which can provide insights into what attacks are (not) being caught by current security mechanisms. At the universities we worked with, these reports

are generated when a user alerts IT security to a compromise, or because existing alert generation mechanisms (including third-party breach alerting services the universities subscribe to) flag an account. In both cases, an IT analyst manually inspects existing logs to attempt to determine if a compromise occurred.

To make use of these compromise reports, we add to the measurement service the ability to accept such logs, anonymize them by encrypting all usernames (using the same key as above), and transfer the resulting data to the persistent database. A similar approach can be used in other deployments of Gossamer to incorporate other relevant data sources, such as logs from MFA services.

3.4 Password-Derived Measurements and Security Analysis

Gossamer enables analyses based on the passwords submitted during login, which will improve our ability to characterize user login behaviors. Passwords, however, cannot be made available to analysts for security reasons. We therefore design Gossamer to only collect limited, useful statistics about the submitted passwords without storing passwords persistently. However, even just statistics computed over user passwords could leak information about the password, so care must be taken on which statistics are persistently stored and made available to the analysts.

In this section, we discuss how we assessed the security implications of the Gossamer logs storing different kinds of password-derived measurements.

Password-derived measurements

To assess risks related to password-derived measurements, we adopt an iterative, simulation-based methodology. We consider a potential logging

schema, namely the set of measurements that we log about login attempts. For a candidate schema, we perform a simulation-based risk analysis that consists of (1) defining a threat model including log exposure; (2) determining a baseline attack that does not exploit exposed logs; (2) developing a log-exploiting attack that incorporates information leaked via fields from the candidate schema; and (3) running simulations using leaked passwords to assess the increased success rate of the log-exploiting attack over the baseline. This allows quantifying risk, and if it is too high we adjust the logging schema and repeat the process until we are satisfied that risk is relatively low.

First we identify potential fields to include in a schema. Figure A.1 in Appendix A lists the fields we ultimately utilize in Gossamer. We also considered several other fields that we eventually discarded as too risky, as we now explore.

To understand password strength, we consider including a zxcvbn strength score [35], and whether or not it belongs to one of the popular password guessing lists (the most frequent 5,000 passwords in RockYou[55] or the top 5,000 passwords generated[3] by Hashcat [57]).

To understand how often breached passwords are submitted, we consider marking passwords as being in well-known, public breaches, which makes those users vulnerable to credential stuffing attacks. For this we use a dataset of 1.3 billion breached username-password pairs [58] and the Compilation of Many Breaches (COMB) containing 3.2 billion pairs [59] released in February 2021. For each attempt, we logged whether the username, the password, or the username-password pair appeared in this breach dataset. We also consider vulnerability to credential tweaking attacks [60, 61, 62] that target passwords similar to a user's other breached passwords. We therefore consider recording the submitted password's edit

[3]We use the rule list `best_64.rule` [56] (a rule list compiled by the Hashcat community of what are considered to be the best 64 rules) with RockYou to generate the guesses.

distance, PPSM similarity [60], and Pass2Path similarity [60] from each breached password for the given username, if it is present in the breach. We also consider logging the edit distance of the submitted password from previous passwords submitted for that username and IP in the same 24 hour window, which would shed light on whether users are making typos or submitting distinct passwords, and what types of password-guessing strategies attackers employ.

Security analysis of measurements

We now turn to making risk assessments about candidate measurements schemas and, in particular, how exposure of Gossamer logs using a candidate schema can be exploited to improve password guessing attacks.

Threat models. As discussed in the last section, we designed Gossamer and our deployment procedures to limit the risk of illicit access to Gossamer logs, but the principle of defense-in-depth suggests that we consider when these mechanisms and procedures fail. For example, an insider attacker could leak the logs to the public internet, or a smash-and-grab attack could somehow compromise the analysis service and exfiltrate the logs.

We therefore consider two threat models. Both threat models assume the attacker obtains a copy of Gossamer logs, can re-identify[4] usernames within the dataset, and seeks to infer the password associated to some particular username. In the first threat model, the attacker can mount an online guessing attack by querying the login service. In the second threat model, the attacker is assumed to additionally have access to some salted hash of the password and so can perform an offline guessing attack. The only difference between the two threat models for our purposes is

[4]Recall that the persistent database contains masked usernames, and it is not exactly clear how attackers would re-identify in this setting. Nevertheless, we conservatively assume that re-identification is perfect.

the expected guessing budget q, which would be on the order of tens to thousands in the online case and hundreds of millions in the offline case.

Looking ahead, it will also be material whether or not the targeted username appears in an attacker-known password breach. If not, the best strategy is for the attacker to modify a target-agnostic password guessing attack (e.g., a dictionary attack) to incorporate relevant information leaked via the log file. In the targeted guessing threat model, the username appears in data breaches known to the attacker, and so the best strategy is to modify a targeted password guessing attack (e.g., [60]) to incorporate relevant information leaked via the log file. We detail particular attacks more below.

In each threat model we consider also a baseline attack (specified below) which performs either targeted or untargeted guessing without exploiting the log files.

Dataset for simulations. The simulations discussed below are based on the breach data used in prior work [60, 63] containing 1.3 billion username-password pairs. There are 370 million unique passwords between length 6 and 30, associated with 1.12 billion usernames. We removed passwords shorter than 6 characters and longer than 30 characters as done in [60, 63]. We split this data so that the attacker has access to 80% to inform a *guess list*, and we randomly sampled 10,000 passwords with replacement from the remaining 20% as target user passwords that the attacker is trying to guess.

Password strength measurements. We first focus on four of the password-derived measurements: (1) whether the password is in the top 5,000 RockYou passwords (RY), (2) whether it's in the first 5,000 Hashcat-generated passwords (HC), (3) the binary zxcvbn (ZB) score of the password that we explain below, and (4) the raw zxcvbn [35] score of the password for comparison. By default, zxcvbn returns a password strength between 0 and 4. We hypothesized that this level of granularity would leak too

much information about the password and speed up a guessing attack. Therefore, we also consider a modified zxcvbn score, where we record 0 if the zxcvbn score is 0, and 1 otherwise; we refer to this as the binary zxcvbn score (ZB). We also consider a combined measurement of all three measurements described above (except the original zxcvbn score) — that is, whether the values of all three measurements match between two given passwords.

Each of these password-derived measurements can be represented as a function $M(w)$ that outputs the result of the measurement as a boolean, an integer, or a tuple. Given a measurement value $m = M(\tilde{w})$ about a randomly chosen target password $\tilde{w}$, the attacker wins if they can guess the target password within q guesses. This is also called the q-success rate (λ_q). We measure λ_q for different values of q. An attacker, given a measurement $m = M(\tilde{w})$, can filter its list of guesses W to only the passwords $w \in W$ that match the measurement, i.e., $m = M(w)$. We let λ_q^M be the success rate of this attack. The baseline success rate λ_q^0 is given by the recovery rate of the attack that simply queries the attacker guess list in descending frequency order (without filtering). Thus, we measure the increase in attacker success as $\Delta_q(M) = \lambda_q^M - \lambda_q^0$.

The results of our simulations are shown in Figure 3.2. We first discuss the online context, where $q \leqslant 1000$. Among different measurements, revealing the zxcvbn score provides the largest improvement in attack efficacy, enabling an attacker to guess 1.6% more passwords in less than 1,000 guesses, compared to the baseline (of 1.8% passwords). The binary zxcvbn score reduces the guessing advantage by a modest amount. Overall, all three measurements combined — RockYou top 5,000, Hashcat top 5,000, and the binary zxcvbn score — would enable an attacker to guess $\Delta_{10^3} =$ 3.1% more passwords compared to not having access to the password-derived measurements. Note that this is without any password policy (except the minimum length requirement of 6 characters).

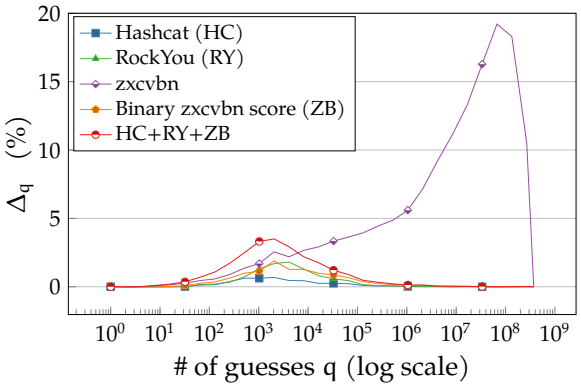


Figure 3.2: Relative improvement in password guessing success rate (Δ_q) due to access to password-derived statistics from Gossamer over the baseline in $q \leqslant 10^8$ guesses.

We also check the success rates for passwords containing at least three character classes (uppercase, lowercase, digits, and symbols), a common requirement. Both universities have three character classes as part of the password policy for current students, staff, and faculty, although old alumni accounts may not satisfy this requirement. When looking at passwords that meet this policy, only 0.8% of passwords are guessed within the first 1,000 guesses without any password-derived information, and the combination of password-derived fields brings this percentage up to just 1.3% ($\Delta_{10^3} = 0.5\%$).

Next we discuss the success rate of an attacker for a large guessing budget $q \in [10^3, 10^9]$. As before, the zxcvbn score can be damaging to the privacy of user passwords, resulting in as high as a 20% increase in attacker success. The binary zxcvbn score provides less information and never leads to more than a 2% increase in attacker success even with a very high number of guesses. Combined measurements also lead to a bounded increase in the fraction of passwords cracked by the attacker, who can guess at most 4% more passwords with the measurement information in an untargeted attack compared to an attacker without the information.

More importantly, most of the improvement occurs for passwords guessed in less than 10,000 guesses because the statistics allow an attacker to rule out many popular passwords when guessing already vulnerable, weak passwords.

Thus we conclude that while including full zxcvbn would be too risky, the increase in attacker success when given the other password strength measures (with the binary zxcvbn score) is sufficiently small.

Edit distance measurements. Other measurements we consider are the edit distance from breached passwords for the appropriate username and the edit distance from other submitted passwords, including the correct ones. Early versions of Gossamer recorded the precise edit distance; we instead now suggest quantizing to just indicate whether a submitted password is within edit distance two of a breached password or other submitted password. Our current implementation does so, and we quantized the data in previously gathered logs.

To come to this conclusion, we observe that an attacker in our threat model can check whether a username is in a breach. (Recall that we conservatively assume the attacker can perfectly reidentify usernames and that they have access to all the breach data used by the measurement system). If the username is not in a breach, then the attacker can proceed as above through a general guess list. If it is, then the attacker can mount a targeted credential tweaking attack in the following way. They start by generating guesses using the state-of-the-art credential tweaking attack based on pass2path [60], seeding it with the passwords in the breach for the targeted username. They can then use the edit distance fields in the log data to filter this guess list by removing any guesses that are the incorrect edit distance from the breached passwords, or not within the appropriate quantized edit distance from the breached password, depending on which schema option we are evaluating.

To assess the improvement in attacks, we compare the modified attack

to a baseline one that just applies pass2path. For simulations, we randomly selected 10,000 username-password pairs from the 20% test data described above, but conditioned on the usernames also appearing in the 80% leaked dataset. We exclude cases where the password is the same on both sides of the split (such passwords would be easily guessed via credential stuffing). For each target username-password pair, we give the attacker the edit distance of the target password from all the breached passwords associated with that username. Then λ_q^{ed} is the success rate for pass2path with guesses filtered to only those that have matching edit distance information. The baseline attacker's success λ_q^0 is vanilla pass2path's success rate.

The baseline success rate in the online setting is $\lambda_{10^3}^0 = 21.6\%$. With precise edit distances knowledge, the attacker can instead achieve $\lambda_{10^3}^{ed} = 85.2\%$. This is an uncomfortably large jump in attacker's success. Quantizing to just edit distance $\leqslant 2$ yields a 22.5% success rate, just a 0.9% increase over the baseline. For larger query budgets (relevant in offline cracking attacks), the improvement for quantized edit distance is even less, at 0.25% increase over baseline for $q = 10^8$.

Discussion. Our simulations suggest that including even moderately granular data such as zxcvbn scores or edit distances in log files might be a risk factor in the case that persistant logs are somehow leaked to adversaries. Therefore we suggest a conservative approach and select logging schemas that avoid improving guessing attacks significantly. contribution of this work, as it allows reasoning in a structured way about risk of password-derived fields.

One current limitation of the framework is that it focuses thus far on attacks against a single user, and so we do not yet know how best to assess the risk of measures capturing similarity of passwords across usernames. Future work could look at extending the framework to look at multi-user attacks. Another limitation is that we rely on best-known attacks (such as pass2path), and as such future work could yield improved attacks. It

is therefore important to retain the ability to sanitize or delete older logs should new results surface previously unforeseen risks.

A full list of measurements logged by Gossamer can be found in Appendix A Table A.1.

3.5 Deploying Gossamer

We partnered with security engineers at two large universities to deploy Gossamer and collect data, beginning in December 2020. We collected data for seven months at U1 and three months at U2 (a shorter timeframe due to the preferences of the IT department). This timeframe encompassed mid-semester, exam, and break periods, so we were able to observe different levels of activities. Throughout this timeframe, we occasionally made updates to some of the measurement mechanisms; these updates were done after careful review of the code by pulling from a git repository accessible to the virtual machine running the measurement service, .

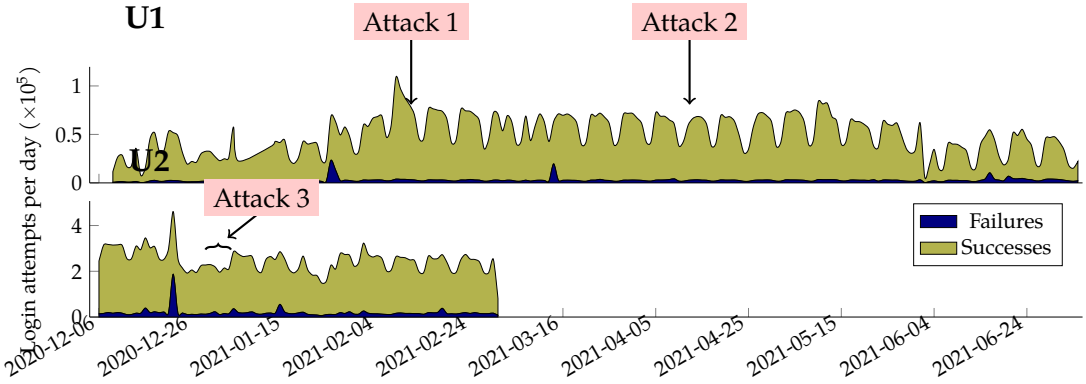


Figure 3.3: Successful and failed login attempts per day at two universities (for a total of 196 K unique users at U1 and 309 K at U2). Potential high-volume attack campaigns we discovered are also shown.

Review process and ethical considerations. Although our research could help understand characteristics of password submissions received by lo-

gin systems, we must also consider the privacy and security risks of inadvertent exposure of the sensitive information in the logs. Our design of Gossamer, therefore, strives to balance these risks with the benefits of protecting user accounts. We conducted a nearly year-long design and implementation effort that entailed a number of external review processes to help guide our research study and reduce potential privacy and security risks to users.

As noted in Section 3.3, Gossamer logs do not include PII. Researchers (with one exception mentioned below for compromise reporting) do not have access to usernames or email addresses. They do have access to IP addresses from where login requests originate.

To protect sensitive data, Gossamer uses a layered approach to security with an encrypted disk, strict firewall rules, and MFA login to access the analysis service. Moreover, Gossamer never stores plaintext passwords on disk; it instead stores a set of password-derived measurements in the persistent database for analysis. Even in the case that these are somehow leaked the chosen measurements represent little additional risk of password disclosure (see Section 3.4).

The research group had ground rules for handling the data, including minimizing granularity of information shared outside the confines of analysis systems, restricting persistent database access to only a subset of researchers, and setting clear expectations about (in)appropriate use of access (e.g., prohibiting re-identification attacks attempting to identify a user from the obfuscated data).

We also went through a careful vetting and approval process with university leadership and their information technology (IT) security departments. This involved presenting to the university leaderships about the goals, design, and procedures associated with the measurement studies, and working closely with our universities' security engineers to design and implement Gossamer. Satisfied with our process and the potential ben-

efits to university account holders, we received approval from university leadership for the deployment of Gossamer.

Before deploying the system, we submitted our study designs for IRB review at each university. The study protocols at the two universities were slightly different in how we report to IT security compromised user accounts. At U1, researchers never had access to plaintext usernames, and the security engineers handled the decryption of reported (encrypted) usernames. Therefore, we received an *IRB exemption* at U1, which found that the research study does not qualify as human subjects research. At U2, security engineers requested that we report the plaintext usernames for operational simplicity. One researcher decrypts the username before reporting compromises. Therefore, we received *IRB approval* at U2, finding the study as a minimal risk human subject research. We did not seek consent from individual users, as we do not know their usernames or email addresses. Instead we obtained explicit approval for conducting this study from the universities' leadership and IT departments who provide the login services. Such waivers of explicit consent from participants were used in prior work (e.g., [64]) and are discussed as an acceptable approach in the Menlo report [65].

Deployment configuration. As mentioned in Section 3.3, Gossamer consists of two services — a measurement service and an analysis service. For U1, we used Amazon EC2 in a virtual private network to host the measurement service as recommended by the U1 security engineers, and we used an on-premise dedicated server for analysis. For U2, we used two separate on-premise virtual machines for running the measurement service and the analysis service. The persistent storage is hosted inside the VM running the analysis.

Strict firewalls were set up for all machines (on-premise or EC2) that block all incoming and outgoing requests except from inside the private network of the respective universities. Only a subset of the researchers have

	Univ. 1	Univ. 2
Session Statistics (with a 360s threshold)		
Avg. session size	2.25	2.21
99th percentile session size	10	6
% abandoned sessions	5.47%	1.96%
User Statistics		
# of unique usernames seen	196,424	309,801
# of valid users	177,286	169,774
# of active users	130,695	110,476
% valid users w/ weak passwords	0.03%	0.06%
% valid users w/ username in breach†	5.79%	3.27%
% valid users w/ passwords in breach†	2.92%	9.34%
% valid users w/ user-pw pair in breach†	0.01%	0.15%
% valid users w/ tweaked password	1.22%	0.66%
% valid users who may be using password managers	24.76%	27.34%
Avg. devices per user per day	1.51	1.91
Avg. devicesper user (over whole time period)	14.51	14.97
Avg. num unique passwords per user	1.96	9.59

† Statistics related to breach data were calculated for data beginning Jan 27 '21 after we added more breach data to the instrumentation.

Figure 3.4: Summary statistics of login requests recorded by Gossamer at U1 and U2. More statistics can be found in Appendix A Table A.3.

access to these machines via SSH, and the access requires two-factor authentication. All incoming and outgoing network connections are logged by the firewall and regularly checked by the security engineers for signs of intrusion attempts. Relevant security patches are checked regularly and applied immediately. The persistent storage uses a MySQL database, which uses TLS for all communication and the underlying disk is encrypted at rest. The login server and the measurement service also use TLS with pinned certificates [66] for all communications. All passwords used are longer than 12 characters, randomly generated, and stored in a password manager. The security engineers also ran scans on the code of Gossamer and the VMs to check for known vulnerabilities.

	Univ. 1	Univ. 2
Login Statistics		
Avg. Login requests per day	49,302	246,274
Avg. # of submitted usernames per day	24,822	61,798
% of requests succeeded	94.99%	92.35%
Avg. # requests per day per user	1.99	2.05
% of requests from mobile device	31.00%	35.57%
% IPs from VPNs, proxies, or Tor nodes	22.08%	4.91%
Submitted password statistics		
% req. w/ password in breach†	2.71%	0.10%
% req. w/ user-pwd pair in breach†	0.07%	0.01%
% failed req. containing a typo	29.67%	12.04%
% failed req. (with edit dist msmt) containing a typo	62.39%	58.37%
% failed req. from mobile device containing a typo	38.63%	38.36%
% failed req. (with edit dist msmt) from mobile device containing a typo	72.69%	81.87%
% pwds tweaked	0.92%	0.34%

3.6 Login Statistics, Patterns, and Observations

We collected data at U1 for seven months and at U2 for three months starting in December 2020. Overall, we observed 10 million requests at U1 and 24 million requests at U2. We show the daily successful and failed requests in Figure 4.4. On average, 5–8% of requests failed; however, on a few days we observed a spike in the failure rate ($> 50\%$). These were high volume attacks that we discuss below. After removing the noise caused by these attacks, we found that users submitted login requests on average 1.99 times per day at U1 and 2.05 times per day at U2. We see fewer login requests at U1 because some login requests are handled via a different authentication server that is outside of our instrumentation.

We saw 196 K unique usernames submitted to U1, out of which 177 K were valid usernames, of which 154 K users had a successful login at least once during our instrumentation period. At U2, we saw 170 K users with at least one successful login during our instrumentation period and 15 K additional users who tried to log in with a valid username but could not complete login due to errors. We consider a user *active* if they have at least one successful login every month. We found about 130 K active users at U1 and 110 K at U2.

Requests originated from 526 K unique IP addresses (approximately 88 K per month) at U1 and 513 K (171 K per month) at U2, and they were associated with 44 K unique user agents at U1 and 31 K at U2. We also used the GeoIP2 database published by MaxMind [67] to find location information for IPs; the majority of login requests at both schools originated from within the United States, followed by China and India. Summary statistics that we report on these login requests can be found in Table 3.4, with additional statistics in Table A.3 in Appendix A.

Characteristics of high volume attacks

We observed three high volume attacks during our instrumentation. Since we are focusing on understanding the full picture of user behavior, we first report on these attacks and then remove them from the dataset to avoid skewing other statistics we report. In total, we removed 54 K requests at U1 and 81 K requests at U2.

Attack 1: *Naïve, multiple-IP, high-volume credential stuffing attack campaign at U1.* Over January 25–26, 2021, four IPs conducted a credential stuffing campaign consisting of 36 K attempts to 19 K users. Two of these IPs were identified by the MaxMind GeoIP database [68] as coming from NordVPN, one from Microsoft, and one from Inwi Mobile [69]. All IPs were active in non-overlapping time periods and submitted up to 100 requests per second. More than 99% of requests from these IPs in this time frame had null user agents. Almost a third of the attempts (29%) of submitted username-password pairs from these IPs were directly from prior breaches, and 60% of submitted passwords were present in prior breaches.

The attack campaign successfully compromised 23 accounts at U1, all of which had been flagged by security engineers and had their passwords scrambled to prevent access. We observed some duplicate username-password pairs submitted across IPs; thus we hypothesize that the attacker used an automated script that iterates through an unfiltered list of breach

data using a variety of IPs.

Attack 2: *Credential stuffing attack using Sentry MBA tool against U1.* An IP hosted in Google Cloud [70] executed another credential stuffing attack at U1 on March 14th, 2021 from 18:42 - 22:22 UTC. This IP submitted over 17 K requests to 15 K unique usernames, submitting approximately 80 requests per second. Of these attempts, 22% of the submitted username-password pairs and 56% of the submitted passwords were directly from the breach data we used with Gossamer. The attacker successfully guessed the passwords for 14 users. Among those, 13 were already recorded as compromised by security engineers; we reported the remaining encrypted username to the security engineers as a potentially compromised account.

Although both Attacks 1 and 2 were credential stuffing campaigns, we suspect that the respective attackers were using two different sets of breach data, as there was little overlap in the users targeted.

We noticed that the attack traffic in this campaign was evenly split between five distinct user agents that were not present in the rest of our data; these five user agents are the default user agents for a tool called Sentry MBA [71]. Sentry MBA is a credential stuffing tool where the user can specify a list of usernames and passwords, a config file for specifying the HTML fields on the target login page, and a list of proxies from which to send traffic.

Attack 3: *High volume, password spraying attack at U2.* On December 22nd, 2020, a total of 12 unique IPs belonging to Digital Ocean Cloud [72] carried out a high volume password spraying attack by targeting 76 K unique users with 169 K requests at an average of 262 requests per minute. The attacker pretended to send requests from `SMTP` and `IMAP` clients. The number of usernames targeted by each IP was evenly distributed among the IPs, and these IPs were active only during the attack period. Less than 3% of submitted passwords were from prior breaches, and none of the submitted usernames were present in prior breaches. We also noticed that all of the

login attempts were for non-U2 usernames, indicating that the breach data was not filtered before the attack. Consequently during this attack campaign, there were no successful logins.

Filtering out attacks. We remove requests from IP addresses corresponding to these three attacks on the respective days for all subsequent statistics to avoid skewing the statistics. Although we did have access to compromised usernames, there was no clear way to determine which IP address compromised a given user. In Section A.2, we show why excluding all IP addresses that contacted a compromised user would not have significantly affected the statistics. This filtering approach works for our setup but may not generalize to other systems. Similarly, although there could be other low-volume attacks that we did not detect, we believe they will not impact the statistics we report.

User and client statistics

Gossamer observed login attempts for 196 K unique usernames at U1. Among the users who could never login, 42 K (21%) of them used a username that does not exist in the U1 login database; however, only 0.1% of these usernames appeared in our breach data. At U2, we saw 310 K users who tried to login, 170 K (55%) that were successful, and a staggering 139 K (45%) usernames that do not exist. We are not sure what caused such a high volume of login submissions with invalid usernames.

More summary statistics on these login requests can be found in Table 3.4 and are elaborated in more detail below.

IP and client characteristics. Requests originated from 539 K unique IP addresses at U1 and 2.47 M at U2. There were 44 K IPs that sent requests to both universities; of these, 621 IPs had no successful logins.

We also recorded the user agent strings present in the HTTP header of the login requests. We observed about 5 K unique user agent strings

at U1, and about 31 K at U2. The top 10 user agents at each school are listed in Appendix A Table A.4. In 22% of requests at U2, the user agent string was set to an empty string; all of these requests were through basic authentication [73]. U1 does not support basic authentication, and less than 1% of the requests had an empty user agent field. We suspect that attempts with an empty user agent field were submitted via an automated script that neglected to set the user agent field. A breakdown of operating systems mentioned in the user agent string appears in Appendix A Figure A.2.

Prior work [74] has suggested that attempts from multiple devices for a user is suspicious and should require additional authentication steps. Freeman et al. defined a *device* as a pair of a unique IP address and a user agent [74]. We observed that on average, 3% of users at U1 and 19% of users at U2 log in from two different devices per day. 11% of users at U1 and 6% of users at U2 have logged from more than fifty devices over the course of the study period.

To find what fraction of IPs were public VPNs, proxies, or Tor exit nodes, we used the Blackbox[5] API. We found that 22% of IPs at U1 were VPNs, proxies, or Tor exit nodes. However, at U1, 16% of all IPs are 10-space IPs, all of which are marked as VPNs/proxies, contributing to this high percentage. These IP addresses are set up by the university IT department and are not accessible outside of the university network. At U2 about 5% IPs were flagged as VPN/proxies by Blackbox API, and 3.6% of all IPs are 10-space IP addresses. Because users may be sharing a VPN or proxy network, it is possible that some users may have the same IP; thus we report most statistics based on device, since it is less likely two different users would also have the same user agent.

The IP address of a user's device might change over time due to DHCP churn or switching between multiple networks. Therefore a login from the

[5]https://blackbox.ipinfo.app/

same device and browser may appear as if it's from multiple different devices, according to the previous definition. So, we conservatively estimate the number of different browsers per user based on the user agent string. At U1, 21% of valid users, and at U2 29%, successfully logged in with ten or more user agents. Thus, many users log in from different browsers, and login security mechanisms must consider this while designing policies.

We were also interested in determining what percentage of users logged in on their mobile versus on a laptop or desktop computer. To do this, we used a regular expression that matched mobile devices on the user agent. We found that at U1, 31% of requests originated from a mobile device; out of those, 84% of requests originated from iOS devices and 16% from Android devices.

These findings may be useful in developing attack detection mechanisms. For example, if a user always logs in from an Android device, it may be suspicious if they attempt to log in from an iOS device.

Password security

The strength and guessability of a password directly affect the security of a user's account. Using the visibility into passwords provided by Gossamer, we investigate password security in terms of strength, the number of unique passwords submitted for a username, and the use of breached credentials.

Password strength. We used four different measurements in Gossamer to measure password strength. We record the bucketized zxcvbn score, as described in Section 3.4; whether the password appears in the top 5k most common RockYou passwords; whether the password appears in the top 5k passwords generated by Hashcat on the RockYou dataset with the `best64` ruleset [56]; and finally, whether the password appears in the top 1000 most common passwords in our breach compilation dataset.

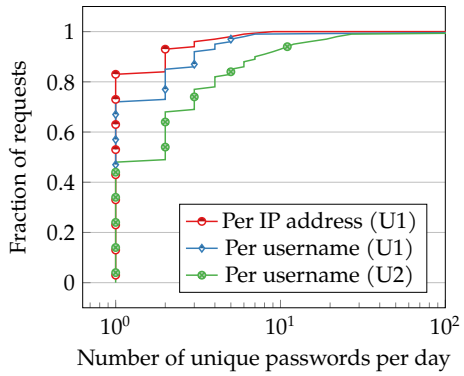


Figure 3.5: Cumulative distributions of unique passwords per username and IP for February 2021. The X-axis is log scale.

Exact percentages of requests matching each of these password strength metrics can be found in Appendix A Table A.3. In summary, we found that the vast majority of valid users were using strong passwords by these metrics. Both universities use strong password policies, requiring a minimum length of 8 and at U1 three different character classes; so this finding is not surprising.

Unique passwords. We also measure the number of unique passwords submitted for a given username or from a given IP address on a certain day.

We observe that a median of one and a 99th percentile of seven unique passwords are submitted against a single username per day. Slightly more unique passwords — a median of one and 99th percentile of nine — are submitted from a single IP address . Both distributions are shown in Figure 3.5. More unique passwords are submitted from a single IP address than for a single username, which makes sense because an IP address may submit to multiple different users, especially if it is a VPN/proxy.

We noted earlier that some organizations may lock accounts that receive a certain number of failed attempts in a given time period. However, these lockout mechanisms do not take into account whether a user submitted

the same password multiple times. Creating a lockout threshold based on the number of unique passwords tried instead of the total number of attempts would improve usability without any improvement in attacker success rates.

To demonstrate this, we consider a lockout threshold of 10 and calculate the number of accounts that would have been locked by counting the number of attempts on a given day instead of the number of unique passwords. We find that 17,863 accounts would have been locked under the simple policy, versus just 2,220 accounts under the policy that counts by unique passwords — an 88% decrease. Similarly for a lockout threshold of 5, implementing the more complex policy would decrease the number of lockouts by 91% from 280,360 to 23,919.

Such a lockout policy could be implemented relatively simply by storing the password hashes submitted for a given user for the last day. Then a lockout policy would check the number of unique hashes submitted in the designated time period. This new policy would improve usability with only a slight increase in implementation complexity and no benefit to a potential attacker.

Breached credential use. To measure the usage of breached credentials, Gossamer records for each attempt whether the username, password, or username-password pair appeared in our breach dataset. Usernames were stripped of a domain name, if applicable, before performing the match. We find that nearly 6% of valid users at U1 and 3% of valid users at U2 appear in our breach dataset, indicating that they have appeared in some data breach in the past. Additionally, we find that 3% of submitted passwords at U1 and 0.1% at U2 appeared in a breach; finally, 0.07 % of username-password pairs at U1 and 0.01% at U2 appeared in a breach. Most of these were failed attempts, but we find that 23 users (0.01% of valid users) at U1 and 254 users at U2 (0.15% of valid users) were still using a breached password as their actual password.

These users are vulnerable to credential stuffing attacks, and this motivates the deployment of breach alerting tools in login systems that would prevent users from continuing to use breached passwords.

For each attempt, we also check for tweaked passwords by recording the similarity of the submitted password to a breached password using three metrics — edit distance, PPSM similarity, and pass2path rank [60]. We define a password as being *tweaked* if the edit distance is less than or equal to two, the PPSM similarity is zero (indicating that they are similar), or the pass2path rank is less than or equal to 1,000. We found that at U1, 0.92% of all passwords submitted were tweaked, and 2,164 users (1.22% of valid users) were using a tweaked password. At U2, 0.34% of passwords submitted were tweaked, and 1,125 users (0.66% of valid users) were using a tweaked password.

However, due to the design of the system, we can only determine whether a user has a tweaked password if they had a previously breached password in our dataset to which we can compare. When we take this into account, we find that nearly 7% of users at both universities with at least one breached password were using a tweaked password. A recent study by Pal et al. [60] reported a slightly higher rate of 8.4%.

We hypothesize that users may append a single character to their old, breached password, causing them to be vulnerable to credential tweaking attacks, in which an attacker tries close variants of breached passwords in an attempt to guess user passwords. Implementing a breach alerting system such as Might I Get Pwned [36] or using a personalized password strength meter [60] in the password change flow could alert users when they attempt to change their password to one that is vulnerable to credential tweaking attacks.

Password changes. Neither of the universities have any periodic password change policies. At U2, users are recommended to change their passwords twice a year, but this is not enforced. New passwords are recommended

to not be the same or similar to any previous passwords, but this is not enforced either. we are able to estimate a subset of the password changes made using the password's edit distance from the previous submissions for a user logged by Gossamer. Because previous submissions are cleared every 24 hours, though, we can only use this measurement in instances when the user logged in with their old and new password on the same day. We find 9,011 total password change events made by 2,893 unique users — — at U1. At U2, we saw 42,827 total password change events made by 7,812 unique users — 211 of which appeared in the compromise database. Of the password change events at U1, 100 resulted in a new password that was in our breach dataset, and 125 resulted in a new password that was a tweaked version of one in our breach dataset. At U1, at least 1.9% of valid users changed their password at one point during the seven month measurement period (about 1.3 K per month) and at U2, at least 4.6% (14 K per month). This is only a lower bound, since we can only measure a fraction of password changes, and prior work is consistent with this bound, reporting higher password change rates [46, 32].

Usability

A longstanding complaint about passwords is their usability. Users often have trouble remembering strong passwords, and they may commit typos especially if they do choose a stronger, more complex password [75, 76, 33]. A key benefit of Gossamer is that it provides a new observation point for measuring password-based login usability.

Login sessions. A user can retry logging in if a login attempt fails (probably due to submitting an incorrect password). To better understand a user's pattern of login retries, we define a *login session* as a sequence of login attempts to a username from the same *device* ending either in a successful attempt or in a period of inactivity (indicating that the user has given up after a series of failed attempts). We define a *device* as a tuple of

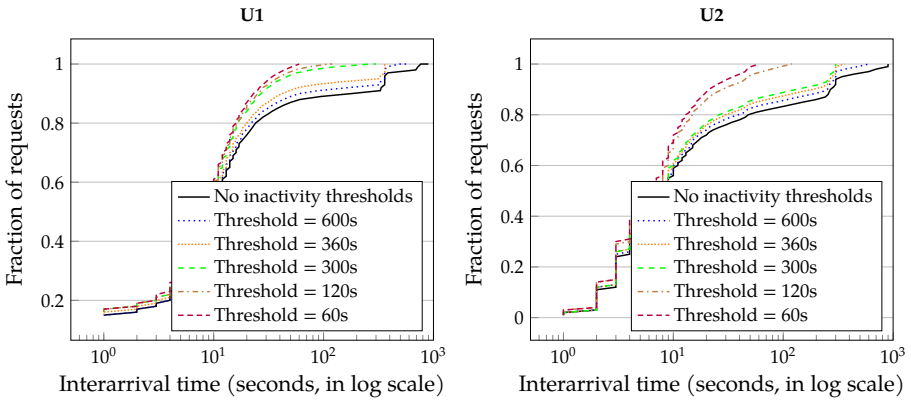


Figure 3.6: Cumulative distribution of interarrival time between requests for different inactivity thresholds for February 2021. The X-axis is log scale.

an IP address and a user agent. It is possible that more than one user may use the same internet connection and thus have the same IP address. In this case, it's possible that they may appear in the same login session if they both submit login requests to the same username in a period of time and share the same user agent as well; however, this seems an unlikely scenario. The definition of login session is parameterized by the length of the period of inactivity, which we call the *inactivity threshold*. We refer to a session that did not end in a successful login as an *abandoned session* and the number of login attempts in a login session as *session size*.

We examine the time between successive login requests, termed as interarrival time, to determine the inactivity threshold that provides stable session size. Because a successful login request indicates the end of a login session, we specifically investigate pairs of successive login requests where the first request was not successful. We show this distribution in Figure 3.6 at both universities for the month of February 2021, as a representative month for which the data collection periods overlap. We limit the X-axis to 15 minutes for easier viewing. At U1, 81% of successive attempts are executed within 60 seconds of the previous attempt, 90% of requests are

Inactivity threshold (seconds)	**Average session size** Univ. 1	Univ. 2
60	2.16	2.44
360	2.29	2.22
600	2.32	2.99

Table 3.1: Average session sizes (number of attempts per session) for different inactivity thresholds

executed within 361 seconds (6 minutes), and 95% of requests are executed within 13 minutes.

We can also see by looking at the average session size for multiple thresholds (Table 3.1) that the choice of inactivity threshold does not significantly affect the session size. Thus, we choose 6 minutes as our inactivity threshold for future statistics involving sessions, since 90% of successive attempts occurred within that window.

With this definition, we find that 51% of sessions at U1 required more than one attempt, and nearly 5% of sessions were abandoned. For U2, 38% of sessions required more than one attempt, and 2% of sessions were abandoned. With so many sessions requiring more than one attempt, there is much room for improvement in usability of password-based logins, which we elaborate on in our discussions on password typos and lockout thresholds.

Retries. The session size indicates how many retries were required before a successful login or the user giving up. Thus we show the average session sizes for different inactivity thresholds in Table 3.1. Some login systems have a lockout policy, in which they lock a user's account after a certain number of failed attempts have been made. In this case, the 99th percentile of session size is 10 attempts per session, providing empirical support for a standard choice for lockout threshold.

We measure the number of sessions per user in a single day, as this will inform how often a user needs to go through the login process, how

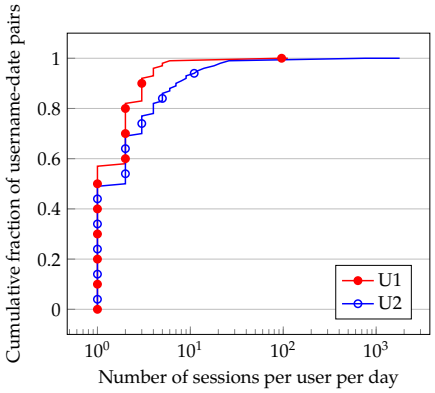


Figure 3.7: Cumulative distributions of number of sessions per user per day for an inactivity threshold of 360s for February 2021. The X-axis is log scale.

important it is to have a frictionless login process, and how effective single sign-on is. We find that a user attempts to log in a median of one session per day and a 99th percentile of six sessions per day; we show the distribution of the number of sessions per user per day in Figure 3.7. However, the tail end of the graph shows that some users attempted up to 112 sessions in a single day. Given that the 99th percentile is six sessions per day, this is probably indicative of suspicious behavior, and in future work this metric may be used in conjunction with the other metrics we've reported to further investigate possible attacks.

To investigate the password-based login usability of mobile devices, we compare the session size and frequency of sessions per day for mobile and non-mobile sessions. We break mobile devicesdown even further by comparing iPhone and Android devices; these distributions are shown in Figure 3.8. We can see from these graphs that users of iOS devices tend to require more attempts than Android ones.

Password typos. One of the areas of friction in a login system may be password typos, especially for stronger, more complex passwords. With the ephemeral datastore in Gossamer, we computed whether a password

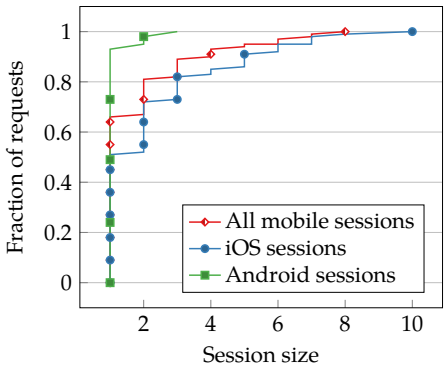


Figure 3.8: Cumulative distributions of session sizes (number of attempts per session) for mobile devices for February 2021. The X-axis for U2 is limited to 10 for comparison with U1 (although the maximum session size consisted of 54 attempts).

submission was within edit distance two of the actual password.

We can estimate the number of password typos using this measurement; however, we only have this measurement for users that later logged in successfully on the same day, which is only 45% of all failed attempts at U1. Thus we find that at U1, 62% of failed requests where we have this measurement contained typos of edit distance two or less, or 30% of all failed requests. In a study at Dropbox, Chatterjee et al. [33] estimated the number of typos to be at least 9% of all failed requests.

Requests originating from mobile devices tend to contain typos even more frequently. Out of all failed mobile requests with the edit distance measurement at U1, 72% were typos (or 39% of all failed mobile requests).

When investigating sessions with two or more attempts, we found that 12% of eventually successful sessions of size two or greater initially failed because of a typo. The remaining failures may be explained by user memory errors (using the wrong password). These findings further underscore the utility of password managers, since they help avoid both typos and memory errors.

Password managers. Although we cannot identify users with password

managers with certainty, we can find the number of users with a certain number of successful login attempts and no failures, and we conjecture that this may be an approximation. For example, we found 25% of valid users at U1 and 27% at U2 have at least ten logins and never had a failed login over the course of our measurements.

Duo logs. Both U1 and U2 have introduced the use of Duo two factor authentication [77] to further strengthen account security. At U2, we were able to analyze Duo logs for successful login attempts and combine them with Gossamer logs to explore the tension between usability and security with respect to MFA. Unfortunately, our logs and Duo log entries do not share any unique identifiers. Instead, we try to match each successful login attempt to a corresponding Duo push within two minutes of the login request originating from the same user. If there were multiple matches to a single successful login attempt, we chose the Duo push closest in time to the login request.

At U2, out of 15.9 M successful login requests, 62% were using Basic Auth [73], which does not require users to enroll in two-factor authentication. Among the remaining 6.0 M successful logins, we could match 89% of requests, and the remaining 11% already had a previously obtained Duo cookie which remained active for 12 hours after a successful authentication. Among the Duo pushes we could successfully match with a login attempt, we found that 96.7% were successful, 3.2% were denied, and only 46 ($< 0.01\%$) were marked as fraud by users. Users on average took 14 seconds to mark their Duo push as valid, slowing down logins for honest users.

3.7 Conclusion

In this chapter, we designed, built, and carefully deployed Gossamer, a framework for securely recording statistics about login requests and submitted passwords during login. Our approach combines system security

mechanisms, a simulation-based approach to assessing risk of different measurements, and procedural mechanisms to enable new kinds of measurement studies. In studies conducted at two large universities in collaboration with their IT security teams, we were able to gather first-of-their-kind measurements about login behavior that shed light on usability, security, and attacker behavior.

4 DISCOVERING AND CHARACTERIZING PASSWORD GUESSING ATTACKS IN PRACTICE

Summary: Remote password guessing attacks remain one of the largest sources of account compromise. Understanding and characterizing attacker strategies is critical to improving security, but doing so has been challenging thus far due to the sensitivity of login services and the lack of ground truth labels for benign and malicious login requests. We perform an in-depth measurement study of guessing attacks targeting two large universities. Using a rich dataset of more than 34 million login requests to the two universities as well as thousands of compromise reports, we were able to develop a new analysis pipeline to identify 29 attack clusters—many of which involved compromises not previously known to security engineers. Our analysis provides the richest investigation to date of password guessing attacks as seen from login services. We believe our tooling will be useful in future efforts to develop real-time detection of attack campaigns, and our characterization of attack campaigns can help more broadly guide mitigation design.

Acknowledgements

Araña was jointly developed by researchers from UW–Madison and Cornell Tech and published at USENIX Securely conference 2023 [12]. In particular I would like to thank and acknowledge the contributions of Dr. Marina Sanusi Bohuk in helping me designing Araña's attack campaign discovery piepline. Dr. Bohuk also evaluated Araña on Gossamer logs at Cornell Tech, and analyzed the resulted discovered attack clusters.

4.1 Introduction

Remote password guessing attacks are one of the most effective and prevalent causes of account compromise for password-based authentication systems [78, 79]. Password guessing attacks are easy to mount for attackers, who may attempt logging in under known usernames with widely popular passwords or perform credential stuffing by submitting username, password pairs that have appeared in previous breaches. Despite being straightforward, such attacks can nevertheless be highly damaging. As a result, the most advanced login services in practice use proprietary mechanisms in an attempt to detect malicious logins using more than just the correctness of the submitted password, e.g., via user risk profiles [79, 74, 80].

Improving such mechanisms requires understanding attacker behavior as observed by login services—a delicate task given the sensitive nature of login data, especially passwords. As such, no in-depth measurement studies of attack behavior as seen from login services have been conducted.

Recently, in our prior work [11], we designed a new login service instrumentation tool called Gossamer. It securely records information about login requests, including certain carefully chosen statistics about the passwords used in the requests. Gossamer was deployed for seven months at one university (U1) and three months at another (U2). We analyzed various aspects of user login behavior, but were only able to identify three obvious, high-volume attacks. How to find and characterize more attacks was left as a tantalizing open question.

In this work, we make progress on answering this open question. To do so, we obtained approval—from our university IRBs and IT security teams—to perform a fresh analysis of the datasets generated by the previous study. These datasets include a rich amount of information on over 34 million login requests, Duo two-factor authentication (2FA) logs (just for U2), and more than 2,000 compromise reports.

Even with the measurements provided by Gossamer, discovering attacks represents a tricky bootstrapping problem. The compromise logs do not implicate specific login requests and, more broadly, there is no ground truth anywhere in the dataset for attacks. This makes the supervised machine learning approaches used in prior work [74, 80] inapplicable to our setting. At the same time, the scale of the problem renders manual analysis prohibitive, and so prior work only used a simple, known method for detecting attacks: Flag IP addresses that individually flooded the login service with requests. This of course misses many attacks.

We develop a new analysis pipeline that we call Araña. Crucially, it focuses not on individual requests but sets of login requests emanating from the same IP within a day. This ensures sufficient signal about client behavior for patterns to emerge. We then built an application-specific filtering, clustering, and manual analysis approach. We filter out likely-benign login request sets using custom heuristics, such as filtering out IP addresses exhibiting high login success rates. We then use an unsupervised learning approach, specifically agglomerative clustering [81], with a custom distance function tailored to login sets. This helps us identify clusters of login sets that can be manually analyzed with little effort to discover *attack campaigns*—sets of login requests likely to be submitted by the same attacker.

Araña is effective at finding these attack campaigns, including coordinated attacks involving multiple IP addresses and those spread out over long stretches of time. We use it to discover and characterize a diverse set of 29 attack clusters. Many are high volume credential stuffing attacks that submit on average one password per targeted username and exhibit a notable fraction of username, password pairs appearing in breaches. In addition to the more simplistic stuffing attacks that quickly flood requests from a handful of IP addresses, Araña allowed for discovering widely distributed attacks that may originate from hundreds of different IP ad-

dresses. Some of these attacks try to be "low and slow", submitting a low number of requests per day at a slow rate. This confirms that credential stuffing attackers attempt to evade countermeasures that focus on individual high volume IPs. These credential stuffing attacks are unfortunately effective: We uncover hundreds of potential successful logins by attackers for usernames not flagged already in compromise reports prepared by security engineers.

We are also able to discover lower volume, targeted attacks by focusing on Araña-identified clusters that exhibit a large number of requests to individual usernames. For example, we identified an attack campaign made up of two clusters that targeted 127 users with on average 25 guessed passwords per user. These attacks included successful logins, suggesting this targeted strategy can also work for attackers.

We discuss many other attack campaigns in the body. Overall, our analyses highlight a number of important takeaways for authentication system designers. First, they suggest that, perhaps unsurprisingly, credential stuffing attacks remain a primary vector for account compromise. This, plus the fact that most compromised users in our study also had passwords in known breaches, underscores the urgent need for broader use of breach alerting APIs (e.g., [82, 63, 79, 83]).

Second, we saw a large number of attacks on Microsoft's basic authentication endpoint at U2. It is the least protected of all of U2's authentication services, as it does not support rate limiting or Duo 2FA. It is also easy for attackers to target any organization using basic authentication by just changing the destination URL. We observed that attackers regularly target such weak points, underlining a challenge for large organizations with heterogenous authentication infrastructure.

Third, mechanisms that look for a large number of submissions per unit time from an IP or to a username are ineffective against many attacks already being deployed in practice. A better approach would be to work

towards operationalizing mechanisms like Araña to detect in (near) real time distributed and sneaky attacks. However, future work will be needed to explore how to make such mechanisms robust against evasion by future adversaries.

Summary. Our contributions include the following:

- We perform the first in-depth analysis of a dataset including over 34 million login events generated by a recent system called Gossamer, in order to discover and characterize remote password guessing attacks.
- We design an analysis framework, called Araña, that shows how to filter and cluster Gossamer logs to enable easy manual analysis and identify attack campaigns.
- We use Araña to discover and characterize 29 attack clusters against two major universities that compromised hundreds of user accounts in total.
- We identify key characteristics and patterns of attacks received by authentication systems at these universities, and we discuss how authentication systems should evolve to counter such threats.

Our work has already had some practical impact in terms of discovering new attacks. We have worked with security engineers from the two universities to perform responsible disclosure of potentially compromised usernames. We believe that Araña will be useful in developing more robust attack detection methods, and we release it as a public, open source project [84].

4.2 Related Work

Passwords remain the most widely used mechanism for user authentication, despite efforts to move past them [85]. We focus our discussion on the literature most closely related to our topic: characterizing password guessing attacks as seen by login services.

Measurement studies on user passwords. User behavior with respect to password choice has been extensively researched. Many studies simulate login services (e.g., via Mechanical Turk) to perform user studies [42, 44]. Others look at password breach data to characterize aspects of user password selection [38, 62]. A handful of studies have measured user passwords in real deployments [46, 49, 85, 86].

Most recently, the Gossamer system [11] was used to measure not only legitimate user password strength, but also various user password submission behaviors as observed by the single-sign on (SSO) login services at two large universities. This chateper uses the same measurement datasets as reported in chapter 3, but focuses on characterizing malicious login behavior.

Password guessing attacks. The literature discussed so far concerned itself with legitimate user behavior, trying in part to assess whether users are selecting passwords that resist various types of guessing attacks. A traditional focus has been on *offline password cracking attacks*, which occur when an attacker attempts to crack password hashes found in exposed password hash databases using tools like Hashcat [57] and John the Ripper [87]. Researchers have also developed natural language processing techniques to improve guess generation [39, 38, 41, 37, 88, 40].

Breached username, password pairs—either obtained via offline cracking, gathered via phishing or malware, or stolen from another web services storing plaintext password—can be used in *credential stuffing attacks*, where an attacker tries to log into a system using breached username,

password pairs. Users frequently reuse passwords across multiple services [62, 61, 60], making credential stuffing one of the most prevalent forms of account compromise attacks [79, 78]. In an effort to reduce credential stuffing attacks, breach alerting services such as HaveIBeenPwned [82], Google Password Checkup [89], and Cloudflare's exposed credential checks [36, 83] provide APIs that check whether a user's passwords have been compromised.

Credential stuffing is one kind of *online guessing attack*—an attack that remotely attempts to log into a service with guessed credentials. Other examples of guessing attacks include *password spraying attacks* that submit a few very popular passwords against a large number of user accounts and *credential tweaking attacks* [60] that submit slight variants of breached passwords.

Detecting malicious logins. Resisting online password guessing attacks requires determining which login attempts are malicious. As Bonneau et al. [90] discuss, login services increasingly should treat login as more of a classification problem, taking into account more than just the correctness of a submitted password. But only a handful of prior works have focused on how to do so.

Freeman et al. [74] were the first to report on a statistical framework using the client IP address and user agent to differentiate between valid and invalid login attempts; this study used real-world LinkedIn login data, but did not include password-based features. They also do not report on observed attack campaigns. Schechter et al. [91] build a malicious login detection system that also utilizes password-based features, including differentiating password typos from other failures and using privacy-preserving data structures such as a binomial ladder filter for detecting frequently used passwords. Their study used simulated data to argue the system's efficacy at detecting malicious logins. Finally, Thomas et al. [79] studied underground forums and tools used to steal credentials—their

only measurements using logins are used to report on the lack of evidence of targeted guessing attacks that try multiple queries against accounts.

The Gossamer chapter [11] reports on three attack campaigns that are easily detected using known techniques (i.e., a huge number of requests from an IP in a relatively short amount of time). Such techniques will only catch obvious attacks and are blind to "low and slow" attacks that purposefully use a small number of guesses per target account (low) and per unit time (slow), or distributed attacks that perform guesses from many different IP addresses. While missing these attacks does not appear to affect the results on benign user behavior reported in [11], understanding attacker behavior requires finding and investigating these attacks.

In summary, no prior work has characterized the behavior of password guessing attacks as seen from login services.

Two-factor authentication. Although effective at preventing account compromise, two-factor authentication (2FA) has yet to achieve widespread adoption [92, 93], and user friction is still very high [94, 95]. As we show, many older accounts at universities are not enrolled in 2FA; and more importantly, even when 2FA is used, recent attacks have shown that attackers can spoof push-based 2FA and hide spoofed pushes by sending them soon after the victim has logged in [96]. Regardless of whether 2FA is bypassed in an attack, we would like to know when an attacker successfully guesses a user's password so that IT security personnel and/or the user can take preventative steps such as changing the password for the indicated account and for any other account that uses the same (or a similar) password.

Unsupervised techniques for attack detection. In this work, we therefore develop a new approach for detecting and characterizing password guessing attack campaigns using Gossamer logs. As we explain in Section 4.4, we did not have success using supervised machine learning approaches

and so instead focus on unsupervised techniques, which have a long history of use in attack detection. For example, they are frequently used in intrusion detection systems (c.f., [97]) and for various kinds of anomaly detection (c.f., [98]). To the best of our knowledge, no prior work has developed mechanisms to detect whether password-based logins are malicious.

4.3 Gossamer Logs

In this work, we use a dataset of login requests compiled from two universities (U1 and U2) via our prior work Gossamer [11] presented in Chapter 3. For completeness of this chapter, we provide some details about Gossamer and how the resulting datasets were collected. We refer readers to chapter 3 for more details.

Gossamer logs. In the previous chapter 3, we introduced a new, privacy-preserving instrumentation approach for use with login systems [11]. The resulting system, Gossamer, provides a secure way to collect measurement statistics about login requests, including a subset of HTTP headers, source IP address, target username, success or failure of a login request, and carefully chosen measurements on the submitted password. Gossamer uses two levels of storage to provide extra security for particularly sensitive information. The submitted passwords are encrypted and stored in an in-memory database. They are cryptographically erased every 24 hours, providing a good trade-off between the ability to calculate password statistics over one-day windows (e.g., the edit distance between two submitted passwords) and the ability to protect the secrecy of passwords even in the low-likelihood case of a full compromise of the instrumentation service. The system also deterministically encrypts the usernames to preserve their privacy and blind researchers from them.

Gossamer was designed in close collaboration with the information tech-

nology (IT) security teams at two large universities. We received approval from the IRB and IT security teams to deploy Gossamer for 7 months at U1 and for 3 months at U2. In total, we collected more than 34 million login requests from 347 K valid usernames. In Chapter 3, we detailed the design of Gossamer and reported on an analysis of this dataset in terms of characterizing legitimate user behavior. At the time we were only able to detect a few obvious attacks that stood out due to their high rate of requests, and left finding more attacks as an open question. This chapter is a first step at addressing this open question.

Duo logs. Both universities use Duo 2FA for most login requests; users are prompted to provide this second factor after they successfully submit the password (and if they do not have a valid "Duo cookie" stored in their browser). The datasets we received include sanitized Duo logs for U2 only.

Unfortunately, there is no identifier in the Duo logs that can be used to uniquely associate a log entry with a particular login request. We therefore had to use a timestamp-based heuristic similar to that used previously in [11], to find the Duo prompt that likely corresponds to a successful password submission. Using the timestamp and encrypted username associated with the Duo request, we correlate the Duo requests with login requests within a 2-minute time window. Out of these requests, 96.7% were successfully completed, 3.2% were denied, and only 46 ($< 0.001\%$) were marked as fraud by the user.

Compromised user logs. At both universities, the security analysts have processes for identifying and reporting compromised accounts. These processes include user reporting mechanisms and third-party breach notification services (e.g., U1 uses a Microsoft product to detect accounts that are sending spam). We were given access to compromised account logs for the period of our previous measurement study, containing the encrypted username, the timestamp reported, the estimated timestamp of the at-

tack (only available at U1), and the reason for compromise. Importantly, compromise reports at U1 were logged from multiple authentication services, but the Gossamer deployment only instrumented the main one (U1 web login); so the number of compromised accounts will be an upper bound for those that were actually compromised through U1 web login. In most cases, analysts respond to compromise reports by scrambling the passwords of compromised accounts to prevent further damage.

Our prior work using Gossamer logs contains some basic details about the compromise logs; we elaborate more here since it will be relevant to interpreting some results later in the chapter. We considered compromised reports for the measurement time period at each university, plus one additional week (which allowed time for analysts to enter compromise reports for attacks that may have occurred during the measurement period). We give summary statistics on this compromise database in Figure 4.1. On average, 190 usernames (0.59% of all valid usernames) are reported every month at U1, and 163 usernames (0.28% of all valid usernames) at U2. At U1, 323 usernames were reported multiple times, 32 of which were reported for two different reasons. Users who are reported twice are compromised on average 26 days apart.

We discuss a breakdown of the reasons for compromise in more detail in Appendix B.1. In summary, we found that the majority of accounts were reported as compromised through large-scale automated attacks. At U1, the estimated time of attack is also reported for each compromised user, and so we investigated the time it takes for an attack to be recorded in the compromise database. We found that 17% of compromised accounts are reported within the first hour after the attack, and 90% of compromised accounts are reported within 61 hours. However, the "timestamp of attack" field is an approximation of the time of attack based on the analyst's best guess. There is no such field recorded at U2.

We also approximate the time of attack by taking the last successful

	U1 (7 mo.)	U2 (3 mo.)
Total reports	1,818	611
Unique usernames		
- reported	1,468	489
- more than once for same reason	323	122
- more than once for diff. reasons	32	0
% of unique usernames compromised / month	0.59%	0.28%
# IPs tried to log in		
- with a compromised username	4,633	6,409
- with multiple compromised usernames	716	156
Max. comp. usernames associated with an IP	382	24
Avg. usernames associated with a single IP	1.98	1.05

Table 4.1: Summary statistics on the compromised accounts reported at each university during the measurement period.

login for a given username before that username is entered into the account compromise database. In doing so, we find that only 4% of compromised accounts at U1 are reported within the first hour after their last successful login, and 90% are reported within 46 days. This large difference shows that either the analyst's estimated time of attack was not very accurate, or the last successful login before a compromise report is not a good approximation for the time of an attack—an important challenge in using compromised logs for detecting attacks, as we discuss in the next section.

4.4 Towards Detecting Attack Campaigns

We now turn to the challenge of discovering remote password guessing attacks using Gossamer logs. Prior work has used supervised machine learning approaches to flag likely malicious login requests [74, 91]. However, they relied on simulated logins, for which they marked each request as benign or malicious.

For login requests observed in practice, flagging each as either benign or malicious represents a challenging bootstrapping problem. Not only is the number of login requests received too massive for a reasonable set

of security engineers to manually analyze, but also there is no clear set of criteria to flag a login request as malicious. Moreover, even if an account has been flagged as compromised by a security engineer or reported as compromised by a user, there is no obvious way to correlate this with individual login attempts because, as discussed in Section 4.3, compromise databases do not include information on the specific login sessions.

Individual login requests often do not contain enough information for even a human analyst to determine if they are malicious. Attackers often use automated scripts to send guesses for usernames and passwords from one or more IP addresses over a period of hours, days, or weeks. To identify attack campaigns we will as we explain below.

Login Sets and Features

The main Gossamer logs consist of a sequence of login requests. Each request entry includes (1) a timestamp; (2) the (deterministically encrypted) username; (3) the client IP address; (4) the client user agent string; (5) whether the submitted password is weak (has a bucketized zxcvbn score of zero, as explained in prior work [11]); (5) the edit distances between the submitted password and previous passwords submitted by the same IP or username; (6) whether the request succeeded or failed due to an invalid username or the wrong password; (7) whether the submitted password is a tweaked variant of a breached password known to Gossamer; or (8) whether the submitted password, username, or username, password pair appear in a breach dataset known to Gossamer.

We group login requests based on the client IP address and date it was received. We call this grouping a *login set*, which we denote by $\mathcal{L}$. The set of all $\mathcal{L}$ sets within a Gossamer log we denote as $\mathbb{L} = \{\mathcal{L}_1, \ldots, \mathcal{L}_n\}$, where n is the number of $\mathcal{L}$ sets in the log. Each $\mathcal{L}_i$ contains all login requests received in a day from an IP address. Note that $\mathbb{L}$ defines a partition over all login attempts.

Group	Features	Acronym
Client	IP address ♦	IP
	ISP ♦	ISP
	User agent string ♦	UA
Volume	Total # requests submitted	NR
	Total # unique usernames submitted	NU
	Avg. # unique password per user	AUP
Login timing	Date♦	DATE
	Mean interarrival time	MIT
	Std. interarrival time	SIT
Success rates	Fraction of submitted requests	
	that failed	FF
	w/ invalid usernames	FIU
Password guessability	Fraction of submitted requests	
	w/ a weak pw, zxcvbn$(w) = 0$	FWP
	w/ pw in breach	FPIB
	w/ username in breach	FUIB
	w/ username, password pair in breach	FCIB
	w/ a tweaked pw in breach	FTP

Table 4.2: Features for $\mathcal{L}$ sets that we use for analysis. Nominal features are marked with ♦; all others are numerical.

We define an *attack campaign* as a set of one or more $\mathcal{L}$ sets. This assumes that all logins in a $\mathcal{L}$ set are either malicious or all are benign. Although it is possible that legitimate users might share the same VPN or proxy IP address as an attacker, we expect such scenarios to be rare (and did not encounter them in our analyses). We leave to future work how to differentiate benign and attack login attempts from a single IP.

To determine if an $\mathcal{L}$ set is potentially malicious or benign, we utilize a rich set of features describing a $\mathcal{L}$ set based on Gossamer logs. The full set of features, consisting of four nominal and 12 numerical features, are summarized in Table 4.2. At a high level, these 16 features can be roughly divided into following groups:

- *Client features* include the source IP address and user agent (UA) within the request. We also look up the ISP associated with each IP address (as reported via the MaxMind Geolocation API [67]). These are useful

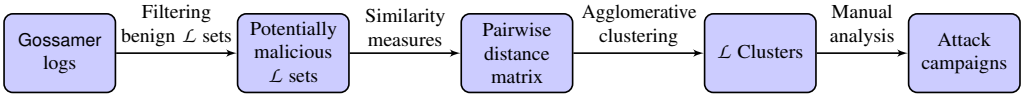


Figure 4.1: Araña's filter, cluster, analyze pipeline for discovering attack campaigns.

for determining whether requests are from the same client device.

- *Volumetric features* include the total number of requests in a $\mathcal{L}$ set (NR), the number of unique usernames targeted in the set (NU), and the average number of unique passwords submitted to a particular username (AUP) for a $\mathcal{L}$ set.
- *Login timing features* include the date of the $\mathcal{L}$ set, as well as the mean (MIT) and standard deviation (SIT) of the interarrival time between requests within the set.
- *Success rate features* measure the fraction of invalid usernames submitted (FIU) and the fraction of invalid username, password pairs submitted (FF).
- *Password guessability features* include the fraction of submitted passwords in an $\mathcal{L}$ set that have a zxcvbn score of zero (FWP), indicating a weak password, as well as the fraction of submitted passwords in a known breach (FPIB), usernames in a known breach (FUIB), username, password pairs in a known breach (FCIB), and the fraction of passwords submitted that are a close variant of a breach password—called a "tweaked password" (FTP).

Together, these features serve as the basis for our attack campaign detection mechanisms.

Initial Attempts Using Compromise Reports

An obvious potential approach for detecting attack campaigns is to utilize compromised account reports to label $\mathcal{L}$ sets as potentially malicious.

However, as discussed in Section 4.3, those reports do not record information sufficient to identify exactly which $\mathcal{L}$ set contains the compromising login attempt. We note that this ambiguity is not just a limitation at the universities we investigated; anytime a login system allows compromise reports from users or third party services, they will not be directly associated with logins. Nevertheless, we experimented with various ways of using compromise reports as ground truth for supervised approaches. We briefly report on two here.

Classifier based approaches. We identified 23,016 $\mathcal{L}$ sets where the corresponding IP address contacted at least one eventually-compromised user account (at any point during the data collection period). Let $\mathcal{C}$ denote the set of all those $\mathcal{L}$ sets. Of course, not all of these are necessarily malicious; but we mark them all as malicious since it is unclear how to label them more precisely.

We first tried to develop a classifier that can identify $\mathcal{L}$ sets likely to have a compromised user based on the features we identified in Figure 4.2. We added to $\mathcal{C}$ an equal number of $\mathcal{L}$ sets not associated with a compromised user account, labeled them as benign, and performed an 80/20 training and testing split on the combined set. We then trained linear regression, decision tree, random forest, logistic regression, and SVM models to predict maliciousness. All the models exhibited very bad precision; the linear regression classifier performed best, with a recall 0.79 but a precision of just 0.13, making it essentially useless. The primary reason is the crude labeling of training data as malicious or benign, presumably including many false positives.

Directed anomaly scoring. We also explored using Ho et al.'s directed anomaly scoring (DAS) [31], tuned via compromise reports, to rank $\mathcal{L}$ sets in order of "suspiciousness." These experiments were promising at detecting new types of attacks, but failed to help us detect attacks spanning

multiple source IP addresses, making it less useful for driving an analysis and characterization of attack campaigns. See Appendix B.2 for details.

4.5 Campaign Discovery Pipeline

We developed an analysis pipeline, called Araña, that combines heuristic-based filtering, unsupervised machine learning to cluster behavior, and a manual analysis review step as illustrated in Figure 4.1. We refer to this as an FCA (filter, cluster, analyze) approach. While the general notion of an FCA-type analysis pipeline is not novel, our application-specific heuristics, similarity measures, and manual review processes are new. Our goal here is not completeness; identifying all malicious logins is impossible. Rather, we build a high precision (low false positive) pipeline that helps us discover a wide range of attack campaigns with minimal manual effort.

We first work to develop heuristics for filtering out likely benign $\mathcal{L}$ sets. First, we remove $\mathcal{L}$ sets that do not exhibit a high failure rate (HFR), and then we remove some anomalous known-benign behaviors (such as misconfigured clients repeatedly making failed requests).

We then use the wide range of features described in the previous section to help us define an application-specific similarity measure $\mathsf{logsetsim}(\mathcal{L}_i, \mathcal{L}_j)$ that outputs a similarity score for any $\mathcal{L}$ set pair $\mathcal{L}_i, \mathcal{L}_j$. This helps us capture the likelihood that two $\mathcal{L}$ sets are part of the same attack campaign. Given logsetsim, we can create a pairwise distance matrix and apply unsupervised agglomerative clustering [81] to discover clusters of $\mathcal{L}$ sets that could each be part of a single attack campaign. The candidates can then be manually inspected by analysts; we report on our findings in Section 4.6.

Filtering Likely Benign Requests

Most logins are benign, and benign requests are a source of obfuscating noise in unsupervised algorithms. We therefore first develop a set of heuris-

tics for filtering out likely benign behavior or, equivalently, identifying potentially malicious $\mathcal{L}$ sets. Our heuristics are based on three assumptions regarding attacker behavior reported in prior work [74, 80, 91]: (a) malicious logins are only a small fraction of the total login traffic; (b) a large fraction of the malicious login attempts fail; and (c) attackers use automated scripts or tools to minimize the time and effort required to send a large number of requests from IP pools (purchased or free proxies, VPNs), hiding their own IP. Based on these assumptions, we first use heuristics to filter out obviously benign behaviors (thereby identifying potentially malicious behavior).

Filtering based on failure rate. We start by focusing attention on $\mathcal{L}$ sets that exhibit a high failure rate (HFR). We choose thresholds for two features: the total number of login requests NR within $\mathcal{L}$ and the fraction FF of these login requests that are failures. We say a $\mathcal{L}$ set meets the HFR condition if $\mathsf{NR} > \mathsf{l}$ and $\mathsf{FF} > \mathsf{f}$ for two thresholds l and f that we set below. Intuitively, benign logins should succeed most of the time, and failure rates are not meaningful for one or two logins.

To choose these thresholds l and f, we first plot the distributions of NR and FF over the subset of $\mathbb{L}$ that includes all $\mathcal{L}$ sets that have at least two login requests ($\mathsf{NR} > 1$) and one failed login request ($\mathsf{FF} > 0$). We show their distributions in Figure 4.2. If we choose extreme thresholds, the HFR heuristic may miss potentially malicious $\mathcal{L}$ sets; but on the other hand, choosing a relaxed threshold may flag benign $\mathcal{L}$ sets as potentially malicious, increasing the noise in our final attack campaign detection procedure.

Thus, at U1, we used the 90^{th} percentile as the threshold, making $\mathsf{l}_{\mathsf{U}}1 = 9$ and $\mathsf{f}_{\mathsf{U}}1 = 0.77$. However, for U2, the 90^{th} percentile yielded $\mathsf{f}_{\mathsf{U}2} = 1.0$, the highest possible value; this is because U2 had more high volume unsuccessful attacks which inflated the 90^{th} percentile. Therefore, we decreased the threshold to the 80^{th} percentile for U2, setting $\mathsf{l}_{\mathsf{U}}2 = 6$

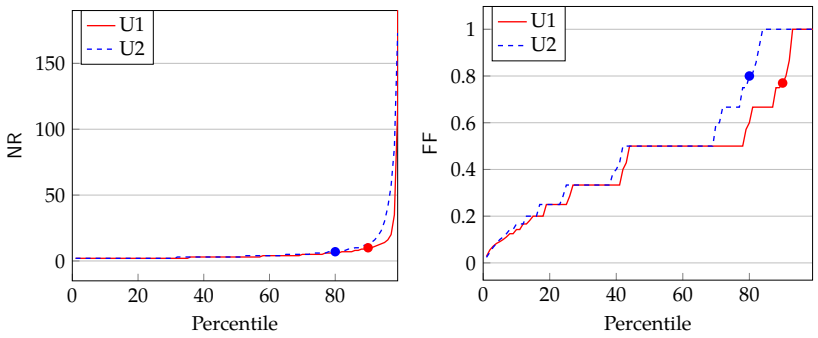


Figure 4.2: Percentile of NR (left) and FF (right) for both universities (shown up to the 99^{th} percentile for viewing). By choosing the 90^{th} percentile at U1 and 80^{th} at U2, we set $l_{U1} = 10$, $l_{U2} = 7$ and $f_U1 = 0.77$, $f_U2 = 0.8$ at the universities respectively.

and $f_U2 = 0.8$. .

Using the HFR heuristic, we filtered out a large number of outright benign $\mathcal{L}$ sets for both universities, as shown in Table 4.3. This allowed us to focus on $\mathcal{L}$ sets exhibiting more anomalous behavior and potentially malicious behavior.

Filtering out benign behavior. After removing $\mathcal{L}$ sets that do not match the HFR heuristic, we further filter out other likely benign behaviors.

First, we filter out $\mathcal{L}$ sets with IPs that have successfully completed Duo requests for all target users at least once on the same day because we assume that a remote attacker conducting large scale password guessing attacks does not have physical access to a victim's Duo authentication device. As we only have the Duo logs at U2, we applied this filtering to the U2 data. We found 563 $\mathcal{L}$ sets matching the HFR heuristic, but all of them contacted only one user and had successfully completed at least one Duo request for that user. Around 90% of these $\mathcal{L}$ sets had submitted $\leqslant 5$ unique passwords, and the failures were due to incorrect password entry—probably just a typo.

Second, we filter out any $\mathcal{L}$ sets having an IP within the university networks or belonging to school proxy/VPN servers. Access to these IPs

Filter	# $\mathcal{L}$ sets	
	U1	U2
$\mathcal{L}$ with NR > 1 & FF > 0	225,468	227,893
$\mathcal{L}$ with NR $> \mathsf{l}$ & FF $> \mathsf{f}$ (HFR)	2,074	11,929
$\mathcal{L}$ with HFR and		
w/ successful Duo request on DATE	n/a	563
IP belongs to school IP pool protected by Duo	60	825
>90% password reuse	419	4,133
Remaining $\mathcal{L}$ sets	1,717	6,408

Table 4.3: Number of $\mathcal{L}$ matching each of the filtering criteria at either university. The last row shows the remaining number of sets after all filtering steps. Threshold values l and f for U1 and U2 are indicated in Figure 4.2.

is restricted, and members of the two universities can only use them after successful authentication and completing the Duo request.

Lastly, we filter out $\mathcal{L}$ sets that seem to emanate from a malfunctioning or misconfigured (benign) client. Specifically, we noticed many IPs submitting a large number of failed login requests with the same incorrect password. Upon further inspection, we observe that these were automated requests belonging to email clients, Outlook Exchange Web Services, and calendar auto-sync agents that were configured with an invalid password. Since any rational attacker would not try the same invalid username, password pair repeatedly, we removed all $\mathcal{L}$ sets reusing the same incorrect username, password pair for more than 90% of their login requests.

After filtering benign $\mathcal{L}$ sets based on the HFR condition, failure rate, successful Duo two-factor submission, school IP addresses, and number of unique username, password pairs, 1,717 potentially malicious $\mathcal{L}$ sets remained at U1, and 6,408 $\mathcal{L}$ sets remained at U2. We show a breakdown of these filtering steps in Figure 4.3.

Clustering Potentially Malicious $\mathcal{L}$ sets

After the filtering step, we cluster the remaining $\mathcal{L}$ sets using the features described in Table 4.2. The key question for clustering is how to define a distance function that gauges similarity between $\mathcal{L}$ sets, such that two $\mathcal{L}$ sets have a small distance if they belong to the same attack campaign.

Similarity modeling. We design a distance function that can measure the likeliness that two given $\mathcal{L}$ sets belong to the same campaign. For the numerical features presented in Table 4.2, we use the normalized difference between two values x and y, defined as $\mathsf{ND}(x, y) = |x - y|/(x + y)$. We chose this particular distance function since $\mathsf{ND}(x, y) \approx 0$ when $x \approx y$, and it is a high value when x and y have a high difference. That said, we believe other numerical distance measures would work well too.

For the nominal features IP, ISP, UA, and DATE, we cannot do a straightforward equality checking to measure their similarity, since these nominal features are sparse. Therefore, similar to Freeman et. al. [74], we use a hierarchical backoff distance (HBD) approach. This technique defines a number of levels: If the two feature values do not match at a lower-level, we "backoff" and use values of the feature from a higher level, while incurring a dissimilarity cost. We set this dissimilarity to $e^{-\ell} - e^{-(\ell+1)}$ for backing off from the lower hierarchical level ℓ to a higher hierarchical level $\ell + 1$, and we accumulate the dissimilarity costs from each level with each backoff.

For the IP feature, we use the hierarchical structure of Internet routing to define four levels. At level $\ell = 0$ we check for strict IP equality; at level $\ell = 1$ we check for /24 subnet equality; at level $\ell = 2$ we check for ISP equality (as reported in the ISP feature); and at level $\ell = 3$ everything is considered equal. Thus the max level for this feature is $\ell^{\mathsf{IP}}_{\mathsf{max}} = 3$. For the UA feature, we use five levels. At level $\ell = 0$ we check if the UA strings are identical. For subsequent levels we extract from the UA string to obtain the application (desktop, mobile, unknown), browser (chrome, edge, other),

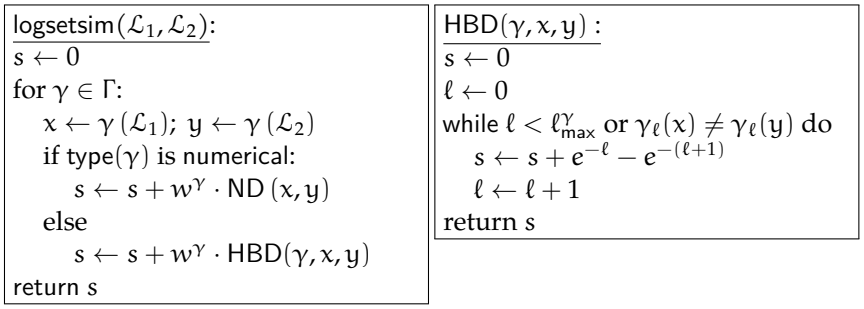


Figure 4.3: Our distance function (logsetsim) to measure the similarity between two $\mathcal{L}$ sets (Left) and the hierarchical backoff distance (HBD) calculation for nominal features (Right).

and OS (Windows, iOS, Mac OS, Linux, other). Equality checks for each of these three define levels $\ell = 1$ through $\ell = 3$. The final level $\ell = 4$ indicates that the user agents do not have anything in common.

For the final nominal feature, DATE, we find the number of days d between two $\mathcal{L}$ sets and compute $1 - e^{-d}$. Thus two $\mathcal{L}$ sets with the same DATE have a distance of 0; with 1 day apart, the distance is 0.63, and so on. Note that this backoff can be calculated (less efficiently) via the same approach as for the other nominal features, by setting $\ell^{\mathsf{DATE}}_{\mathsf{max}}$ equal to the maximum number of days.

Pseudocode for the complete logsetsim and the hierarchical backoff distance (HBD) is shown in Figure 4.3. The set Γ includes all feature we used (shown on Figure 4.2), and we abuse notation slightly by letting $\gamma \in \Gamma$ define a function that maps a $\mathcal{L}$ set to the relevant feature value. For the nominal features, we further let γ_ℓ denote the function that extracts the ℓ^{th} level from the feature value. We also define a predicate type over features that indicates whether the feature is nominal or numerical. We weight the distance values computed for each feature based on a hyper-parameter called feature weight $w^\gamma \in [0, 1]$.

Clustering. We used an agglomerative clustering technique [81] that can

work with a non-metric distance function. Agglomerative clustering starts with each data point as a single cluster and only merges two clusters if their distance is smaller than a given threshold. Adjusting the threshold can help avoid clustering together $\mathcal{L}$ sets showing different login behavior. To set the distance threshold appropriately, we rely on a knee locator method [99] frequently used in clustering algorithms to find the correct threshold for distances. We set the linkage type to "average" and set the distance threshold for U1 and U2 to 0.44 and 0.51 respectively after applying the knee locator method at each university separately. The silhouette score [100] for agglomerative clustering was 0.19 for U1 and 0.17 for U2, which beat other approaches we tried (see Appendix B.3).

Implementation details. We implemented our similarity model logsetsim in 240 lines of code written in Python 3.6. To extract the four hierarchical levels for the IP feature, we used the MaxMind GeoIP database [67]; and for the UA feature, we used the ua-parser package [101]. Given an $\mathbb{L}$ of size n, we computed an $n \times n$ distance matrix to be used for various clustering algorithms. Computing the distance matrix takes $O(n^2)$ time; however, it can be easily parallelized. We used 40 threads and were able to compute the distance matrix for $n = 1,717$ within 9 minutes at U1 and for $n = 6,408$ within 28 minutes at U2 (using an Intel Xeon Linux machine with 56 cores and 125 GB of memory). For the clustering step, we used the sklearn [102] library, and clustering completed in less than a minute for both universities.

Attack Campaigns Discovered

Based on our designed similarity modeling, the agglomerative clustering approach described in Section 4.5 produced 366 clusters from 1,717 $\mathcal{L}$ sets at U1 and 640 clusters from 6,408 $\mathcal{L}$ sets at U2. For each cluster we recompute the feature values mentioned in Figure 4.2, after taking the

union of $\mathcal{L}$ sets. Next, we sample a few of the top most interesting clusters for manually analyzing them.

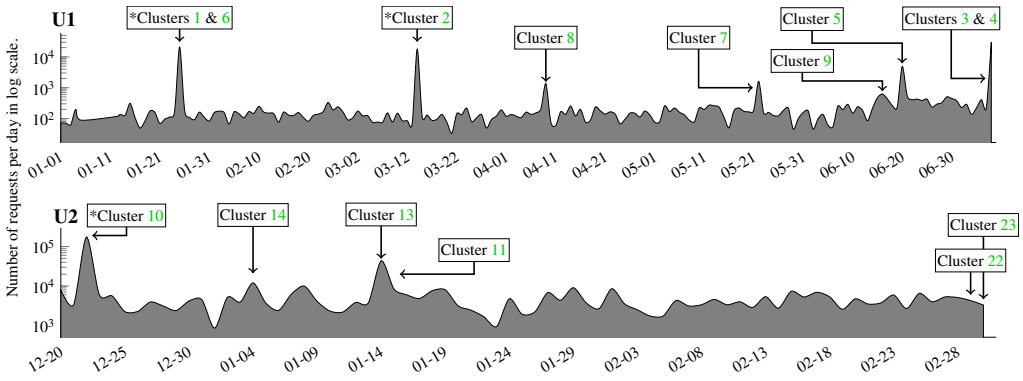


Figure 4.4: Number of requests per day for the 1,752 and 6,408 potentially malicious $\mathcal{L}$ sets at U1 and U2 respectively as shown in Table 4.3. Attack clusters we found in Section 4.4 are shown in red boxes. Clusters marked with a * were identified in prior work [11] as high volume attacks. Note that the x- and y-axes are different for the two universities for better visualization.

To identify the large volumetric attack campaigns, we sampled clusters containing a high number of requests or high number of unique usernames. At U1, we found eight such clusters with $\mathsf{NR} \geqslant 1,000 \vee \mathsf{NU} \geqslant 1,000$. At U2, we sampled 12 clusters with $\mathsf{NR} \geqslant 5,000 \vee \mathsf{NU} \geqslant 5,000$. We chose these thresholds by manually observing that the clusters found after these thresholds do not clearly show malicious behavior. As our goal is to characterize attacks, we focus on precise identification of attack campaigns, foregoing high recall. Thus we sorted possible attacks by a few different metrics. For example, we sorted by the number of unique usernames and found one additional attack at U1; all the top clusters at U2 had already been found using the volumetric thresholds.

All of the above sample clusters did not exhibit any targeted behavior—sending roughly one unique password to each user on average. Therefore, to capture attack campaigns exhibiting targeted behavior, we consider clus-

ters sending a high number of unique passwords submitted per username on average. Although at U1 we did not find any such clusters appearing to be attacks, we discovered eight clusters from U2 that submitted an average of at least 25 unique passwords per user. Altogether, we sampled nine attack clusters at U1 and 20 attack clusters at U2. Attack clusters at U1 sent on average 8,432 requests to 1,294 unique usernames, with a total number of 41 successful logins among these 9 clusters. At U2, we saw an average of 14,358 requests submitted per cluster to 7,614 unique usernames, with a total number of 1,116 successful logins among all 20 clusters. We describe these clusters further in Section 4.6.

4.6 Analysis of Attack Campaigns

As seen in the last section, Araña's FCA pipeline helped us identify 29 clusters that are probable attack campaigns. We first describe a subset of these campaigns in more detail, to better understand attack behavior as seen in practice. In Section 4.6 we generalize from these case studies to identify a variety of observed higher-level attacker behaviors.

Example Attack Campaigns

First we describe some attack clusters representative of different types of attack behavior we observed and show how we group some of them into campaigns involving more than one cluster. We show the timeline of the attack campaigns discussed in Figure 4.4, and the full list of attack clusters we found is shown in Figure 4.5.

Previously reported attacks. Three attacks were manually identified and discussed in prior work [11]. All three were also detected via our FCA pipeline: Attack #1 from [11] corresponds to Clusters 1 & 6, Attack #2

corresponds to Cluster 2, and Attack #3 corresponds to Cluster 10 (also shown in Figure 4.4).

Attack #1 from [11] was a credential stuffing attack distributed over four IPs that were active on different days. Each of the IPs submitted lots of requests very quickly, with a peak rate of over 100 requests per second. As such, these were easily identified manually as related attacks by the similar attack behaviors and time period when they were active. This still required significant manual analysis of Gossamer logs; however, our FCA approach automates this analysis. Two clusters identified by the FCA method capture the bulk of the $\mathcal{L}$ sets associated with these attacks. One $\mathcal{L}$ set containing 16,035 requests that were previously manually identified was omitted; this $\mathcal{L}$ set had a lower failure rate (0.67) than our heuristic filtering thresholds (which was 0.77 at U1).

Attack #2 from [11] was a high volume credential stuffing attack using SentryMBA [103] to send requests from a single source IP address. Since the attack consisted of only one IP address on a single date, the cluster consisted of one $\mathcal{L}$ set, ranked second by volume of requests in our FCA approach.

Attack #3 from [11] was observed at U2 and involved 12 distinct IP addresses with an average rate of 188 requests per second. This attack was easy to detect due to its sheer volume and rate of requests. Manual inspection revealed that the attack requests mimicked SMTP and IMAP clients, which helped in manually clustering them. Again, our FCA pipeline automated this step, placing all activity from these IP addresses into a single attack cluster.

Our new FCA approach helps automatically identify these attack campaigns with little error relative to the manual analysis used in prior work [11]. However, it did split the first attack into two clusters, and it also missed an IP address from that attack. Across these three attack campaigns, 17 $\mathcal{L}$ sets were identified by both the manual and FCA approach as being part

of attack campaigns, and one $\mathcal{L}$ set was identified by the manual approach but not the FCA approach. We show the corresponding confusion matrices in Appendix B.4. Note that the FCA approach also found many attack campaigns hard to manually detect that we describe next.

Future work can look into whether unsupervised clustering approaches can be improved to be more accurate than manual inspection. For now it is clear that FCA helps find and characterize attack campaigns in an automated fashion that can later be investigated by an analyst. We now discuss in further detail the new attacks discovered using our pipeline.

Clusters 5 & 7: Repeated attacks from the same IP. We saw two credential stuffing attacks at U1 (Clusters 5 & 7) a month apart (May 22 and June 20) from the same IP address and user agent (and thus, likely from the same attacker) attempting to login to 4,480 and 1,347 users respectively. The five user agents used in the June 20 attack were a subset of the seven used in the May 22 attack, and the two attacks targeted 295 overlapping usernames. Thus we believe that the IP was under the same attacker's control for both attacks. This IP was active on 24 other days, but only submitted between one and four requests per day to nine distinct usernames over that time period. We believe the attack exhibited credential stuffing behavior, as the fractions of breached passwords submitted were 0.63 and 0.32 respectively, and less than 8% of the passwords submitted were weak passwords. Other features such as the AUP, FTP, and FSP were relatively similar between the two clusters. All but two usernames were incorrect across the clusters, so we believe the attack was curated for U1.

Cluster 12: Multi-day attack from a single IP. Although most of the attacks we saw were finished within a 24 hour period, some attackers may spread out the attack over multiple days in an attempt to stealthily avoid detection. We saw this behavior in Cluster 12 at U2. In this attack, one IP from Microsoft Corporation ISP sent one request per minute on 15 days in a two month period. In total, the IP sent 13,289 requests to 8,192 unique

usernames. There was evidence of credential stuffing, as 61% of passwords submitted appeared in breach data known to Gossamer, and fewer than 0.04% of them were weak passwords. We did not see any evidence of targeted behavior, as there was an average of only one unique password submitted per username; however 63% of the attempted usernames were valid, meaning the attacker curated their password guessing attack for U2. The IP attacked the basic authentication protocol that does not have two factor authentication set up at U2. This IP successfully guessed the passwords of 501 user accounts over this time period, only 163 of which were detected by the security engineers.

We saw similar multi-day attacks in Clusters 15, 18, 19, and 29, albeit for shorter time periods. Such attacks may easily avoid simple volumetric detection methods (as they were missed in the prior work) and thus show the utility of a richer feature-based clustering approach.

Cluster 14: High volume, distributed, credential stuffing. We saw several cases exhibiting high volume, distributed, curated credential stuffing. For example, in Cluster 14 at U2, 843 IPs belonging to 13 ISPs submitted 10,535 requests to 7,771 usernames over the course of 23 hours. Since each IP submitted only around 12 requests, it avoided detection by the manual analysis performed in the prior work [11]. This attack exhibited credential stuffing, with 66% of passwords present in breach data; and 48% of the submitted request contained a valid usernames from U2. During this attack, the involved IPs successfully guessed the passwords of 258 unique user accounts, 74 of which were confirmed by security engineers independently. Clusters 3, 4, 8, 9, 11-13, and 15-19 showed similar signs of high volume, distributed, credential stuffing with varying levels of curation to the target university.

Cluster 17: Possible credential stuffing with unknown breach data. A few attacks showed a high fraction of tweaked passwords (passwords that are a close variant of a breached password for the targeted user), but

a lower fraction of breached passwords. For example, in Cluster 17, we saw three IPs submit 6,290 requests to 2,513 unique usernames over the course of two days. Although 91% of the submitted usernames in this attack campaign are present in the breach data used by Gossamer, only 18% of the submitted passwords appear with those usernames. Of the remaining 82% submitted passwords for those users, we that found 70% are only small tweaks of the password(s) present in the breach data with the corresponding username, and 40% of the submitted passwords exactly match the breach data. This may indicate that the attacker is using a breach dataset that is unknown to Gossamer; but because users choose similar passwords across web services [62], we can still detect these breached passwords not present in Gossamer's breach data.

This attack produced successful logins to 35 users, two of which were independently detected by the security engineers. Cluster 20 showed similar signs of possible credential stuffing with unknown breach data, with a high fraction of tweaked passwords. Neither of these attacks exhibited targeted behavior (having AUPs of 1.55 and 1.56 respectively); therefore, we do not believe these are credential tweaking attacks [60].

Clusters 22 & 23: Targeted attacks. In a few cases, we saw a higher average number of unique passwords tried per username. For example, in Cluster 22, we saw 73 IPs submit 1,878 requests to 75 unique usernames, with an average of 25 unique passwords tried per username, each attacking only one user. Cluster 23 was active on the same day and executed a very similar attack: 52 IPs submitted 1,296 requests to 52 usernames, each submitting an average of 25 unique passwords to one unique username, just like Cluster 22. Thus we believe they are part of the same campaign.

In this attack campaign, the attacker clearly used a popular password dictionary, as 82% of passwords submitted had weak passwords (zxcvbn score of 0). The fraction of valid usernames for Clusters 22 and 23 were 0.96 and 0.88 respectively, indicating that the attack data was probably

curated to the university. The campaign was successful in guessing the passwords for two usernames, neither of which was detected by the security engineers.

Clusters 24-28 exhibited similar targeted dictionary attack behavior, although with fewer IPs and a lower number of total requests. Our clustering approach, however, failed to cluster all of these $\mathcal{L}$ sets together. This is a common limitation in agglomerative clustering [104].

Higher-Level Attack Behaviors Observed

Attack campaigns employ various strategies to maximize their success in compromising user accounts. Broadly, there are two components in an attack campaign that the attacker has to choose: (a) the types of username, password pairs to submit, and (b) how login requests with those username, password pairs are delivered to the target service. We noticed different approaches to each of the two components in the attack campaigns we found at U1 and U2, as we discuss in more detail below. The list of different behaviors we observed as well as some example clusters showing that behavior are shown in Table 4.4. We also explore the geographical distribution of attacks, but relegate details to Appendix B.5.

Types of submitted username, password pairs. A key component in an attack is the set of guessed username, password pairs, as the success of the attack depends on it. There are different strategies for picking username, password pairs—e.g., an attacker can choose usernames belonging to the university and try several popular passwords against those users, or an attacker can choose breached username, password pairs with or without filtering the usernames specific to the university.

Curates usernames to the target university. We found that all attacks at U1 curated their set of usernames, with more than 75% of requests containing a username present in U1. However, at U2, only 7 (out of 20) attack

Attack component	Behavior	Ex. clusters
Type of username, password pairs	Curates to the target university	2, 21
	Targets certain users	22, 23
	Uses breach data	9, 19
	Submits popular passwords	22, 23
Delivery of requests	Distributed among multiple IPs	3, 11
	Distributed among multiple days	12, 19
	Ends quickly (within 24 hours)	1, 11
	Exhibits low interarrival time	5, 11

Table 4.4: Different attack behaviors we observed.

clusters contained more than 50% valid usernames. Clusters 10 and 11 combined submitted nearly 200 K requests, but only 286 of them contained a username present at U2. Even when an attacker attempted to log in multiple times for a particular username, we found that in several cases the username did not exist. Some attacks are therefore rather indiscriminate, trying arbitrary usernames without checking first whether they are valid for the target authentication service.

Targets certain users. An attacker may submit requests against one, a few, or many unique usernames. Among attacks we detected, it was more common for an attack to target a large number of usernames in a "horizontal attack." Most of the attacks we found were horizontal attacks—trying one or two passwords per username but for a large number of usernames. An attack submitting multiple unique passwords against fewer usernames may be exhibiting *targeted* behavior. We found six attack clusters targeting 137 users in total, each with more than 25 unique popular passwords. We do not know if the set of passwords used were the same for different users, as Gossamer logs at U2 do not allow comparing passwords submitted to different usernames.

Uses breach data. In our analysis, we found that attackers often use prior breaches to source their username, password pairs in what is popularly known as a credential stuffing attack. In six out of nine clusters at U1 and

in one cluster at U2, 50% of the submitted username, password pairs are present in the breach data used by Gossamer. Among the remaining 22 clusters, eight clusters (all at U2) had more than 50% of the targeted usernames in a known breach, and all but one attack cluster at U2 submitted passwords that were found in prior breaches. This reiterates the threat of credential stuffing.

Uses popular passwords. Although most attacks used breached passwords, some attacks relied on especially popular passwords. Attackers may use popular password dictionaries [105] curated from prior password breaches for such attacks, and such passwords will, by definition, also be flagged as having appeared in a known breach. We found a number of attack clusters submitting popular weak passwords, such as Clusters 22 & 23. More than 81% of the passwords submitted in these attacks appear in the 1000 most frequent passwords found in the breach data known to Gossamer. Notably, all of these attacks were targeted, submitting more than 25 passwords per user. Both universities have password selection policies that aim to disallow popular, weak passwords. Nevertheless, we found that such attacks were successful in compromising two users at U2.

Delivery methods for attack requests. After choosing the set of username, password pairs to submit, the attacker must decide how to submit them to the target service. Primarily, the attacker must identify how quickly they can submit all the username, password pairs without being detected. To do so, attackers can use multiple IP addresses to parallelize the attacks, and/or spread the attack over a long period of time.

Distributed among multiple IPs. We found several attack campaigns that distributed their login requests over multiple IP addresses. In particular, we identified three large attack clusters using more than 500 IP addresses and five other clusters each using more than 50 IP addresses. In the two clusters with the most IP addresses at U2 (Clusters 11 & 14), the majority

of IPs were flagged as proxies by Blackbox API [106]. By distributing over multiple IPs, an attacker can achieve a higher throughput, as exhibited in cluster 11 which used multiple IPs to achieve a very fast request rate. This distribution of requests across IPs can help to avoid volumetric detection mechanisms, as exhibited in clusters 22 and 23.

Duration of the attack. Attacks can be short-lived and bursty or spread across multiple days. All attack clusters at U1 were short-lived, finishing within 19 hours and four of them completing within five hours. At U2, however, we found that most high volume attacks (submitting more than 5 K requests) span multiple days, and two clusters were active for over 10 and 15 days, respectively. Interestingly, these attacks would be very difficult to detect by looking at their behavior on only one day; however, our clustering approach helps to see the full attack by combining the attack behaviors from multiple days.

Interarrival time. Finally, sending requests too quickly could also trigger alarm or lockout. Therefore, some attackers try to submit requests at a lower rate. However, we see at least four attack campaigns that submitted as much as 60 requests per minute at U1 and U2. These attacks are not curated, meaning that the majority of the submitted usernames do not belong to the target university and thus could not result in a successful login. Both U1 and U2 have a soft rate limiting policy, meaning that too many unsuccessful attempts against a username could lead to account lockout for 15–30 minutes. However, we did not find any attack that submitted requests fast enough to a single user that could trigger that lockout. We are unsure if attackers adapted their attacks to this lockout constraint or if their typical behavior avoids such lockouts.

Endpoints targeted. An attacker may choose an attack endpoint with the least safeguards in place. We found that most of the attacks at U2 targeted the Microsoft basic email authentication endpoint. We suspect that there

are four reasons for this high usage of the basic authentication endpoint: (a) this endpoint does not support two factor authentication; (b) it has poor (non-existent) rate limiting of requests, as basic authentication requires frequent submission of passwords; (c) the requesting IP seems to be from Microsoft (as seen by the authentication server); and (d) attackers would have to make only a small change (the URL) to attack different organizations using basic authentication.

4.7 Discussion

We design Araña, an attack analysis system based on our FCA—filter, cluster, analyze—framework. Using Araña with the logs created by our prior work [11], we identified several attack campaigns in two university login systems. These attack campaigns are spread across 29 clusters as detailed in Table 4.5 and compromised 1,157 users across two universities. Although our study is limited to the attacks received by two universities, we believe the patterns we observed across attacks can form a basis for building future defenses against password guessing attacks.

At U1, the most common type of attack we saw was a combination of a high volume, distributed, curated, short-lived, credential stuffing attack; the interarrival times of the attacks varied from 55 milliseconds to 108 seconds. At U2, almost all the attacks were high volume credential stuffing attacks, but they varied in whether they were distributed, curated to the university, short-lived, or exhibiting a low interarrival time. Of the eight lower volume attacks we found, all were curated, short-lived, targeted credential stuffing; and they varied as to whether they were distributed across IPs. The distributions of attacks observed in the two universities are quite different, possibly because Gossamer gathered data from all login endpoints at U2, but only the main login endpoint at U1. Thus, attackers

utilizing different endpoints at U1 are not reflected in Gossamer logs.

Efficacy of breach alerting services. To determine how much a breach alerting service could mitigate password guessing attacks, we investigated the characteristics of the password from the last successful login before a user was reported as compromised. We found that 2% of compromise reports were associated with a breached username, password pair (that is, the last successful login to that username before the compromise report was a breached username, password pair), 12% were associated with at least a breached password, and 19% were associated with a tweaked password. At U2, we observed that 11% of compromised users' last logins were made with a breached username, password pair, 46% were made with at least a breached password, and 1.51% were made with a tweaked password. Thus automatically resetting passwords of users using vulnerable, breached, or tweaked credentials could have prevented a significant fraction of account compromises. At U1, 71% of eventually compromised users used a breach password at some point during the instrumentation period. This further underpins the need for proactive breach alerting [63] services that could have saved 47% of account takeovers as stated earlier.

Key observations from attacks. Our findings reiterate the ongoing threat of credential stuffing, which has been the most prevalent and successful form of account compromise attack. We also observed a few low volume targeted attacks against specific users. Attackers often use multiple IPs from cloud providers, VPNs, and network proxies to distribute and hide their attacks, but our FCA approach can identify such campaigns even when each individual IP makes only a handful of requests. Such observations should be taken into account while building defense policies. For example, locking user accounts due to a small number of incorrect attempts rarely translates to higher security, whereas discouraging users from reusing passwords from other websites and using breach alerting services can be very effective. Proactive breach alerting [63] using services

such as HIBP [82] would be very helpful in combating credential stuffing attacks.

Using an FCA approach for attack analysis. Our FCA approach helps analyze attack campaigns by clustering $\mathcal{L}$ sets with similar attack behavior. This enables a security engineer to look at the whole attack, instead of considering login activities from a single IP or on a single day. As we show, several attacks are spread across multiple days and use multiple IPs; in some cases, they may use only one IP and spread the attack over multiple days, making them very hard to detect. We believe clustering seemingly unclear behaviors into groups can help security engineers see the attack pattern, detect hard-to-detect attacks, and have confidence in their judgment. We envision that our FCA approach can be used on authentication logs such as the ones produced by Gossamer [11] to group possible attack campaigns and sort by those that are more likely to be actual attacks. Then analysts can manually investigate further, using the features and groupings produced by the clustering to inform their decision. Thus our FCA approach could reduce the effort needed to analyze the complete logs of all IPs; such a tool could prepare daily (or weekly) reports about potential attacks.

4.8 Real-World Deployment Constraints

Despite the many campaigns detected by Araña, there are certain challenges that may occur during deployment in real-world settings. Here we talk about a few that we have faced.

Recording password-derived features. Most organizations do not record password-based features for good reason, and while Gossamer [11] showed how to record them safely, it is still an open question whether widespread deployment of recording password-derived features is wise or necessary.

Araña provides evidence that certain password-derived features can be useful for attack detection. For example, we identified that password breach statistics are very effective at detecting credential stuffing attacks. On the other hand, password similarity statistics—which require storing passwords in memory for 24 hours—are not as effective for attack detection in the FCA approach. So organizations could log a small set of already proven-helpful password-derived features and apply clustering using those features. We detail clustering results without password based features in Appendix B.3 and leave the open question of which features are the most helpful for detecting different types of attacks to future work.

Reporting compromised accounts. Based on the attack campaigns discovered by our FCA approach, we believe 41 unique user accounts were compromised at U1, and 1,116 were compromised at U2 (some accounts were compromised multiple times by different campaigns).

At U1, 37 of the 41 compromised accounts detected by Araña were already detected via other mechanisms, and manually recorded by the security engineers as compromised. We reported the remaining accounts and received feedback that they were indeed compromised, and the remaining account had already been deleted.

At U2, only six out of 1,116 compromised users detected by Araña had been independently flagged by the security engineers. We reported the rest to U2's security engineers in two rounds. In the first round, we reported 823 compromised accounts, 373 (44%) of which the security engineers confirmed as definitely compromised. For the remaining 450 accounts, U2's security engineers said they could not verify fully due to unavailability of adequate logs but that their best guess was that they were compromised based on what logs were available. In the second round, we reported another 293 compromised usernames; however, we received a similar response mentioning that the IT department did not have adequate logs to determine whether these were compromised.

In practice, confirming if accounts are compromised is nuanced and often relies on indirect indicators like an account being used to send spam or a report from an account owner. Elsewhere, compromise status can be inherently ambiguous. Even so, our experience working with the security engineers suggests that more detailed and persistent logging may help. Likewise, our experience with Araña suggests that password-derived measurements can be helpful to analysts when attempting to characterize and confirm compromises. We also note that this difficulty in confirming account compromise makes it difficult to automatically finetune many of the hyperparameters in Araña (e.g., distance thresholds and percentile thresholds for filtering $\mathcal{L}$ sets). Therefore we had to rely on experimentation with different thresholds and manual analysis on password-derived and volumetric-based measurements which took comparatively much more effort. Nevertheless, Araña provides a way to use Gossamer logs to identify high-volume and distributed attacks that were not detected by existing mechanisms.

Robustness of malicious login detection. Although this shows that Araña can detect existing attack strategies, it is unclear if it will remain effective even when the attacker adapts to such defense mechanisms by actively modifying their login requests to evade detection. Furthermore to mitigate the fallout of compromise effectively, one may wish to customize Araña to run it in a timely fashion instead of waiting for months of login data. Even notifications delayed by hours or days can be useful, as they significantly narrow the window of exploitability for compromised accounts. In Appendix B.6, we present a timely version of Araña that can be used to detect attacks within a day of their launch. Additionally how robust the timely version Araña is to evasion attacks using a new data driven approach.

4.9 Conclusion

Using data collected at two universities from a measurement framework for logging password-derived information behavior, we designed a set of features that describe an IP address on a given date, along with a clustering algorithm to group IP addresses active on certain dates together into probable attack campaigns. We describe several of these clusters in full detail to show the differences and similarities between attacks, and we discuss our observations about behavioral patterns of the attacks as a whole. Our results indicate that clustering approaches can aid an analyst in detecting and labeling suspicious groups of requests that may be part of the same attack campaign. They also provide initial progress towads the future design and deployment of real-time, automated attack campaign detection tools.

ID / Univ.	Duration	# IPs	# ISPs	# unique users (NU)	# requests (NR)	Avg. uniq. pwds per user (AUP)	Frac. weak pwds (FWP)	Frac. pwd in breach (FPIB)	Frac. user-name, pwd in breach (FCIB)	Frac. pair in breach (FUIB)	Frac. tweaked pwds (FTP)	Frac. valid usernames (FVU)	failure rate (FF)	Comp. users	Comp. users flagged.*
1§ / U1	5 h	3	3	10,424	18,093	1.21	0.12	0.56	0.41	0.69	0.06	0.94	1.00	1	1
2§ / U1	4 h	1	1	15,209	17,827	1.10	0.04	0.56	0.22	0.42	0.06	0.97	1.00	14	13
3‡ / U1	14 h	555	4	12,659	15,117	1.00	0.01	0.14	0.02	0.58	0.46	0.78	1.00	14	12
4‡ / U1	11 h	77	2	12,318	14,603	1.00	0.01	0.14	0.02	0.58	0.45	0.78	1.00	7	6
5 / U1	41 m	1	1	4,480	4,593	1.03	0.08	0.63	0.21	0.44	0.08	0.97	1.00	1	1
6§ / U1	4 h	3	1	2,101	2,683	1.21	0.07	0.66	0.48	0.69	0.04	0.98	1.00	0	0
7 / U1	9 h	1	1	1,347	1,481	1.00	0.00	0.32	0.10	0.22	0.05	0.99	1.00	2	2
8 / U1	19 h	85	13	894	1,246	1.00	0.01	0.33	0.08	0.92	0.84	0.92	1.00	2	2
9 / U1	37 m	30	7	219	241	1.00	0.06	0.92	0.78	0.96	0.10	0.93	1.00	0	0
10§ / U2	11 h	12	1	76,321	169,573	1.01	0.00	0.03	0.00	0.00	0.00	0.00	1.00	0	0
11 / U2	10 m	663	10	27,488	33,304	1.00	0.10	0.79	0.00	0.00	0.00	0.01	1.00	0	0
12 / U2	15d	1	1	8,192	13,289	1.02	0.04	0.61	0.07	0.54	0.02	0.63	0.93	501	6
13 / U2	3 d	1	1	7,939	12,240	1.30	0.05	0.66	0.10	0.75	0.00	0.35	0.99	120	0
14 / U2	23h	843	13	7,771	10,535	1.01	0.08	0.66	0.00	0.71	0.00	0.48	0.97	258	0
15 / U2	5 d	1	1	4,662	9,714	1.07	0.09	0.84	0.00	0.91	0.00	0.40	0.99	32	0
16 / U2	2 d	2	1	4,934	7,323	1.00	0.06	0.58	0.00	0.23	0.00	0.81	0.84	786	0
17 / U2	2d	3	2	2,513	6,290	1.55	0.03	0.40	0.18	0.91	0.70	0.37	0.99	35	0
18 / U2	10 d	1	1	1,902	5,434	1.37	0.04	0.55	0.27	0.82	0.41	0.39	0.99	13	0
19 / U2	5 d	1	1	3,584	5,261	1.04	0.07	0.76	0.80	0.98	0.18	0.42	1.00	6	0
20 / U2	2 d	3	2	1,756	5,199	1.56	0.04	0.50	0.22	0.92	0.72	0.35	0.98	83	0
21 / U2	23 h	1	1	5,076	5,103	1.00	0.06	0.59	0.00	0.26	0.00	0.80	0.85	777	0
22 / U2	13 h	73	43	75	1,878	25.04	0.82	0.88	0.01	0.31	0.29	0.96	1.00	1	0
23 / U2	21 h	52	38	52	1,296	24.92	0.82	0.89	0.01	0.34	0.32	0.88	1.00	1	0
24 / U2	12 h	4	4	4	101	25.25	0.80	0.86	0.00	0.55	0.50	0.51	0.99	1	0
25 / U2	16 h	3	3	3	80	26.67	0.81	0.75	0.01	0.65	0.58	0.30	0.99	1	0
26 / U2	8 m	1	1	1	27	27.00	0.85	0.85	0.00	0.00	0.00	0.00	1.00	0	0
27 / U2	8 m	1	1	1	27	27.00	0.85	0.78	0.04	1.00	0.96	1.00	1.00	0	0
28 / U2	13 m	1	1	1	25	25.00	0.84	0.88	0.00	0.00	0.00	0.00	1.00	0	0
29 / U2	4 d	1	1	3	468	38.75	0.15	0.59	0.00	0.67	0.16	0.33	1.00	0	0

‡ Since the measurement collection ended during these clusters, some of these measurements (such as NR and NU) may be lower bounds.
§ These clusters are part of campaigns that were discussed in previous work [11].
† Duration units are days (d), hours (h), and minutes (m).
* The last column represents the accounts that were flagged independently by security engineers at the respective universities.

Figure 4.5: Attack clusters detected using a set of heuristics and manual review. The first column notes the ID of each cluster as we refer to them in the chapter. The attack IDs we describe in detail (Section 4.6) are shown in bold font. We discovered a total of 41 and 1,116 unique compromised users at U1 and U2 respectively.

5 DETECTING ATTACKS FROM COMPROMISED PASSKEYS

Summary: FIDO synced passkeys address account recovery challenges by enabling users to back up their FIDO2 private signing keys to the cloud storage of passkey management services (PMS). However, it introduces a serious security risk — attackers can steal users' passkeys through breaches of PMS's cloud storage. Unfortunately, existing defenses cannot eliminate this risk without reintroducing account recovery challenges or disrupting users' daily account login routines. In this chapter, we present CASPER, the first passkey breach detection framework that enables web service providers to detect the abuse of passkeys leaked from PMS for unauthorized login attempts. Our analysis shows that CASPER provides compelling detection effectiveness, even against knowledgeable attackers who strategically optimize their attacks to evade CASPER's detection. We also show how CASPER can be seamlessly integrated into the existing passkey backup, synchronization, and authentication processes, with only minimal impact on user experience, negligible performance overhead, and minimum deployment and storage complexity for the participating parties.

Acknowledgements

Part of the research presented in this Chapter was developed while I was doing a research internship at Visa Research and from our accepted paper at USENIX Security conference 2025 [107]. I would like thank my collaborators Dr. Sunpreet S. Arora and Dr. Ke (Coby) Wang from Visa for their help. Specifically I would like to mention my mentor during the internship Coby who helped me in designing CASPER, security proofs, and model checking experiments to evaluate the effectiveness of CASPER.

5.1 Introduction

FIDO2-based user authentication has compelling security guarantees . While they are promising at (finally) replacing passwords, recovering accounts in the event of device loss remains a significant challenge preventing the adoption of FIDO2. To address this challenge and accelerate the adoption of FIDO2-based passwordless user authentication, Microsoft, Apple, and Google announced *passkeys* [2], which enables cloud backup and multi-device synchronization of FIDO2 private keys. Today, there are many passkey management services (PMS) , e.g., iCloud Keychain from Apple [5], Google Password Manager [6], Password Monitor from Microsoft [7], 1Password [8], LastPass [9], and DashLane [10].

While synced passkeys address the account recovery concern, backing up passkeys to a centralized cloud server poses a serious security risk: attackers can steal users' passkeys through PMS breaches and use them to take over users' web service accounts. The breach can happen when an attacker compromises a user's PMS account (possibly by guessing the account password) or when an insider attacker gains access to the cloud backup storage [108, 109]. Unfortunately, existing passkey backup and synchronization implementations lack the capability to detect unauthorized access to users' passkeys stored at the PMS.

In this chapter, we address this problem by proposing CASPER[1], which, to the best of our knowledge, is the first framework to detect the abuse of passkeys leaked from PMS. Underlying CASPER is a decoy-based detection technique [110, 111, 112]: it hides the *real passkey* within a list of decoy passkeys that are indistinguishable from the real one. As a result, an attacker who steals the passkeys from PMS and attempts to use them to log into the user's account at a relying party (RP), e.g., a website, inadvertently will end up using a decoy passkey with high probability. These login attempts will trigger breach detection at the RP, indicating

[1] CASPER is short for Capturing pASskey comPromise by attackER.

the abuse of passkeys stolen from the PMS. However, unlike focusing on allowing relying parties (RP) to detect breaches of *their own password databases*, as explored in prior works [110, 111, 112], CASPER enables the relying parties to detect breaches of PMS — a different service. To do so, there are additional usability challenges that CASPER must overcome. For example, false detections by a denial-of-service attacker must be avoided, as they would undermine the trustworthiness of any breach detection system. Arguably more importantly, even if an additional user secret is needed in such a system, it should be easy for users to manage so it avoids reintroducing the account recovery problem, and it should not disrupt their daily account login routines.

To meet these requirements, we design a new passkey backup and restoration (BnR) protocol and a compromise detection (CD) algorithm. The BnR protocol protects the real passkeys by encrypting them under a *recovery key* that is retrievable given a user secret (denoted by η, only known to the user). A user with the correct η will be able to recover the real passkey for account logins.

An attacker who could have already obtained users' passkeys from PMS breaches without CASPER deployed must now guess η in order to decrypt and retrieve the user passkeys. Each guess will produce a well-formed passkey, i.e., a well-formed private signing key but not necessarily the *correct* key. As a result, without η, the attacker cannot determine if their guess yields the real passkey or a well-formed decoy unless they test it by attempting to log in with it to the user's RP account. As the decoys are indistinguishable from the real passkey from the attacker's perspective, they will easily end up using a decoy passkey during the login attempt. RPs that implement our CD algorithm will detect such attempts and deem them unauthorized use of passkeys leaked from PMS. This detection will enable RPs to make informed decisions and promptly mitigate such threats.

To show the effectiveness of CASPER, we model sophisticated attackers who strategically optimize their attack strategies to evade CASPER's detection by leveraging useful information leaked from *already compromised* websites where users have accounts and CASPER is deployed. Our detection effectiveness analysis shows that CASPER provides compelling true-detection and negligibly small false-detection probabilities even when the user secret η contains low entropy (e.g., as low as 5 bits) — confirming that CASPER does not impose additional high-entropy secret management burdens on users. Also, CASPER does not impose any extra tasks on users during their daily account login routines. Importantly, as a general framework with η serving as a configurable component, CASPER can be fine-tuned for specific authentication scenarios balancing deployability, usability, and security.

We provide a prototype implementation [113] and show that CASPER can be seamlessly incorporated into FIDO2 authentication protocols (i.e., CTAP 2.0, WebAuthn) without introducing significant storage costs or causing noticeable login delays. As CASPER aligns with FIDO Alliance's ongoing exploration of trusted signals to detect abuse of synced passkeys, we believe our work will spark interest within the authentication community to consider its adoption in practice.

To summarize, our contributions are as follows.

- We propose CASPER, the first framework to detect the abuse of FIDO2 synced passkeys leaked from passkey management service (PMS) providers. Importantly, CASPER can also be easily extended to detect breaches of other cryptographic credentials that are widely used today, such as HMAC-based / time-based one-time passwords (HOTP / TOTP) seeds.

- We demonstrate the detection effectiveness of CASPER systemically against sophisticated attackers who tries to evade detection by leveraging breaches from other already breached websites.

- Through a prototype implementation of CASPER, we confirm that CASPER introduces negligible performance and storage overhead for all parties involved and demonstrate that deploying CASPER requires minimal modifications to PMS and RPs.

5.2 Related Work

The foregoing discussion in Section 2.2 echoes the challenge we are facing today to secure users' passkeys against breaches, revolving around many moving parts. These include, but are not limited to, the security of users' PMS accounts, the resistance of user secrets (for key derivation or escrow) to offline cracking, the correctness of PMS implementations, and PMS's defenses against both remote and insider attackers. The failure of any of these components could expose all of a user's passkeys, leading to users' mistrust in the provider [21] and, more critically, large-scale account takeovers across numerous RPs, which highlights the urgent need for an effective framework to detect passkey breaches from PMS storage, which we provide in this Chapter. Now we will discuss existing work related to this Chapter.

Decoy-based credential breach detection. Existing decoy-based detection systems for user-chosen passwords [110, 111, 112] allow websites to detect the breaches of *their own password storage*. The idea is for the websites to plant a fixed number of decoy passwords, known as *honeywords*, in their password storage alongside the user-chosen password. If an attacker breaches the website's password storage and attempts to exploit the leaked (hashed) passwords to access the user's account at the website, entering a honeyword will trigger a password breach detection.

Traditional honeyword systems [110, 112, 114] rely on the information asymmetry between the defender (e.g., a website or RP) and the attacker — they assume that the attacker cannot access the secrets stored by the web-

site to distinguish between honeywords and the user-chosen passwords. This information asymmetry inevitably requires additional assumptions about what the website knows that the attacker does not in order to enable effective detection. For example, the information asymmetry may arise from a *honeychecker*, which stores an identifier of the user-chosen password that is known only to the website [110]. Alternatively, it may originate from a pseudorandom number generator only accessible by the website [112], or from the assumption that the attacker is unaware of the deployment of a breach detection system facilitated by machine-dependent password hashing [114].

Amnesia [111] is the first *symmetric* design that allows attackers to learn the entire persistent state of a website and enables detection based on probability distribution changes of some password "markings" that can help distinguish between honeywords and the user-chosen one.

Existing honeyword systems mentioned above aim to detect unauthorized logins at a website or an RP — the same party where the password breach occurs. In comparison, CASPER enables the detection of unauthorized login attempts that abuse breached passkeys at an RP when passkey breaches occur at a different party — a PMS provider. For this reason, existing honeyword systems fall short of addressing the problem CASPER aims to solve. To be practical, however, CASPER adopts a symmetric security assumption similar to that of Amnesia and avoids relying on additional security assumptions required by other traditional designs.

Credential keys / password backup storage leakage. Existing work on *preventing* unauthorized access to users' accounts due to breaches of credential backup services can be roughly categorized into two camps.

The first line of work [115, 116] aims to prevent the leakage of users' cryptographic keys by distributing them securely across multiple parties on the cloud. In the context of synced passkeys, one might consider distributing users' passkey storage across multiple PMS providers to mitigate

the risk of breaches affecting one or a subset of them. However, this approach would pose a significant deployment burden, as it is against the centralized nature of existing major PMS providers, e.g., Apple, Google, and Microsoft, which manage users' passkeys within their own ecosystems and is incompatible with their current implementations for credential synchronization and storage.

The second line of work[117, 118, 119] proposes *honey password vaults* to make attackers difficult to perform efficient offline-cracking on leaked *password vaults / managers*[2]. However, the insufficient rate-limiting defense of most websites against online guessing [120, 121] and vulnerabilities in current honey vault designs against sophisticated attacks [122, 119, 123, 124] have cast doubt on the overall efficacy of these proposals in recent years.

In comparison, CASPER is a *detection* framework to enhance users' account security by enabling RPs to detect the abuse of passkeys leaked from PMS. Notably, CASPER is deployable and user-friendly, while remaining effective even against highly sophisticated attackers.

5.3 Threat Model

In this section, we detail the participating parties and our threat model.

Participating parties. Following the convention of FIDO2, we refer to the device that generates the private signing keys as the *authenticator*, and the websites as *relying parties* (RP). We assume the authenticator supports synced passkeys, meaning that it has the capability to synchronize the private signing keys with a passkey management service (PMS) *provider* who stores users' passkeys on their cloud storage. The authenticator itself could be dedicated hardware that is platform-independent (e.g., YubiKeys)

[2]Password vaults / managers are cloud backed up synced passwords of the user — encrypted under a single master password.

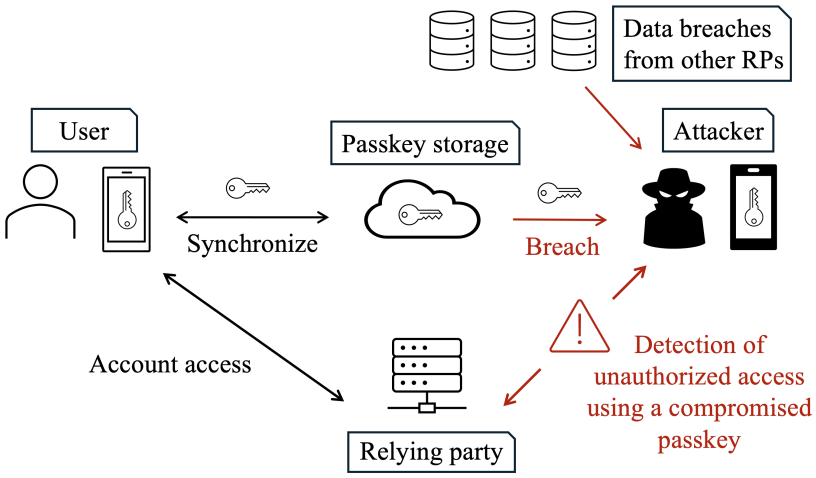


Figure 5.1: Overview of CASPER and threat model: CASPER allows a relying party to detect the abuse of a compromised passkey for unauthorized account access when the attacker obtains the passkey from a breached passkey storage of a passkey management service (PMS) provider and also has access to a certain number of data breaches of other RPs.

or a virtual one that is platform-dependent (e.g., phones or laptops). The users authenticate themselves locally to the authenticator (e.g., via PINs, biometrics) and allow a *client* (e.g., browsers, mobile apps, etc.) to complete account registration/authentication at a RP using passkeys.

Introducing `caat`. We consider a credential backup abuse attacker (`caat`) who has access to the data (and associated metadata, if any) stored at the PMS provider. The goal of `caat` is to undetectedly take over user accounts at RPs where CASPER is deployed. We follow the threat model of FIDO2 authentication [125] and assume that the user and her devices (e.g., the authenticator and client involved) are trusted. We also assume that RPs are trusted for honestly detecting attacks by `caat` and have employed other state-of-the-art defenses such as protection against denial-of-service attempts during account registration and login. All communication channels are assumed to be secure and authenticated using a standard secure

communication protocol (e.g., SSL/TLS [126]).

Making `caat` realistically stronger. Given the increasing number of data breaches that websites are experiencing today, we also consider the case where `caat` may have access to data breaches (e.g., obtained from black market forums [127]) from breached websites where the target user has accounts. We follow, to the best of our knowledge, the strongest threat model [111] in the literature on credential breach detection (as discussed in Section 5.2) where `caat` is given *read access* to the *entire* persistent storage of a certain number of compromised websites, including all data used for account login or registration and breach detection by CASPER. However, like prior work on breach detection [110, 114, 112, 111], we do not allow `caat` to actively compromise those breached websites, and we assume the transient information that arrives in a login attempt is not stored by a breached website and not available to `caat`. Note that the primary focus of this chapter is to detect breaches of PMS storage, and we do not aim to enable breached websites to detect PMS breaches, as we view those data breaches as *static* information that can be leveraged by `caat`. However, our framework can be extended to enable even passively breached RPs to effectively detect PMS breaches by incorporating the probabilistic detection method proposed in [111].

To summarize, as shown in Figure 5.1, we assume that `caat` has access to users' passkey backups at PMS, as well as to the persistent storage of a certain number of passively breached websites. As we will show in Section 5.6, CASPER can effectively detect a sophisticated attacker like `caat` when it adopts its optimal strategy to take over user accounts at *unbreached* RPs and to evade detection by CASPER.

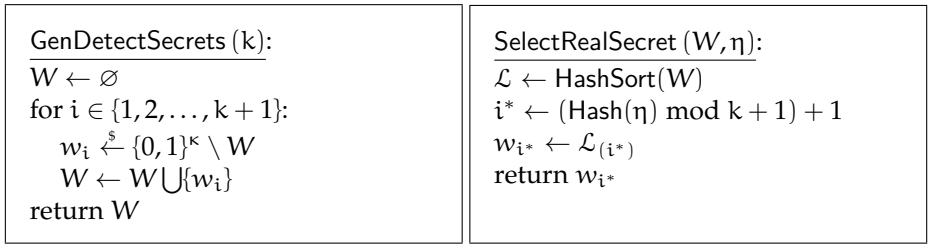


Figure 5.2: An instantiation example of GenDetectSecrets and SelectRealSecret. Here HashSort sorts all elements in W by their hash values seeded by η, i.e., $\mathsf{Hash}(w_i \parallel \eta)$ for all $w_i \in W$, and then outputs these elements as an ordered list. Hash is a collision-resistant hash function. Here i^* indicates the index of the *real* detection secret.

5.4 Detection Secrets

CASPER relies on a set of detection secrets $W = \{w_1, w_2, \ldots, w_{k+1}\}$ where one of these secrets is the real detection secret and its index is denoted by $i^* \in \{1, 2, \ldots, k+1\}$ throughout the chapter. Following this notation, the *real* detection secret is denoted by $w_{i^*} \in W$, while the remaining k secrets in $W \setminus \{w_{i^*}\}$ are referred to as *decoy secrets*.

Survivable user secret η. We use η to represent the information needed to identify w_{i^*} from a given W. This η should not be shared with any PMS providers or RPs, and we assume η to be *survivable*; that is, we assume that η is always accessible by the user (even in the case of device loss or failure). For example, η could be instantiated in practice by secrets that can be easily recalled by the user (e.g., PINs) or retrievable from physical objects or trusted parties (e.g., credit card verification codes, bank account numbers), or biometrics. The options of instantiating η with a proper user secret will be discussed further in Appendix C.5.

Decoy generation. How to generate good decoy credentials [128, 129, 130, 131, 132] has been more of an orthogonal line of research to our work. In this chapter, we consider a pair of procedures $\mathcal{G} = \langle \mathsf{GenDetectSecrets}, \mathsf{SelectRealSecret} \rangle$

Symbol	Description
w / W	detection secret / the set of detection secrets
v / V	passkey verifier / the set of auth. verifiers
V'	the set of *active* decoy passkey verifiers $V' \subseteq V$
s / $\tilde{\mathsf{s}}$	passkey / encrypted passkey
$\mathfrak{u}$	the recovery key used to encrypt s
$w_{\mathfrak{i}^*}$ /$\mathsf{s}_{\mathfrak{i}^*}$ /$\mathsf{v}_{\mathfrak{i}^*}$	the *real* detection secret / passkey / passkey verifier
$\mathfrak{z}$	nonce used to encrypt s
$\mathfrak{k}$	# of decoys, $\lvert W\rvert = \lvert V\rvert = \mathfrak{k}+1$
α	fraction of *active* decoy passkey verifiers $\alpha = \frac{\lvert V'\rvert}{\lvert V\rvert-1}$
$\mathfrak{n}$	# of breaches `caat` has observed
$\mathfrak{m}$	# of relying parties (RPs) `caat` wants to login
sid	unique identifier of the RP
aid / uid	unique user account identifier at PMS / RP

Table 5.1: Notations used in this chapter.

where GenDetectSecrets is a *randomized* procedure for generating decoy secret candidates and SelectRealSecret a *deterministic* detection secret selection procedure. The procedure GenDetectSecrets produces a set W of size $\mathfrak{k}+1$ when given an integer $\mathfrak{k}$. SelectRealSecret takes W and a user's survivable secret η as input, and outputs $w_{\mathfrak{i}^*} \in W$ so that $w_{\mathfrak{i}^*}$ is the real detection secret and other $\mathfrak{k}$ are decoys. A simple instantiation example of GenDetectSecrets and SelectRealSecret is shown in Figure 5.2. As such, when the passkey backups are additionally encrypted under an encryption key derived from $w_{\mathfrak{i}^*}$, the attacker who attempts to recover users' passkeys from stolen passkey backups, without knowing η, can only make guesses on which in W is $w_{\mathfrak{i}^*}$. Such guesses may be undesired in other authentication scenarios, but our framework, as we will show in later sections, leverages such guesses to detect the compromise of users' passkey storage on the cloud.

5.5 CASPER: A New Detection Framework

To enable a relying party (RP) to detect the abuse of users' passkey leaked from passkey management services (PMS) provider, CASPER introduces a new passkey backup and restoration (BnR) protocol (Section 5.5) and a compromise detection (CD) algorithm executed by the RP (Section 5.5). For brevity, we refer to the attacker who has compromised users' passkey storage at breached PMS provider as `caat`. Section 5.5 provides a high-level overview of CASPER and its design considerations. Table 5.1 presents the key notations used throughout this chapter.

Design considerations and overview

Design considerations. Consider that a user (say Alice) uses a PMS provider to backup and synchronize her passkeys for RP account logins. As usual, Alice is responsible for certain operations already required by PMS provider (e.g., account creation with PMS using a master password and new authenticator setup using passcode) for passkey synchronization or access across her existing devices / authenticators. CASPER is designed to effectively detect malicious login attempts by `caat` using Alice's passkeys leaked from a PMS provider, while meeting the design requirements as detailed below.

❶ *Easy to deploy.* CASPER does not require PMS providers to modify their protocol designs or implementations for passkey synchronization and storage, and it requires only minimal changes on the RP side. Also, CASPER is compatible with existing two-/multi-factor authentication or risk-based authentication schemes and does not require any additional (trusted) hardware for PMS providers or RPs. Please see Section 5.7 for further discussion of the deployment considerations for CASPER.

❷ *User-friendly.* To avoid introducing usability challenges, such as the account recovery problem caused by requiring Alice to manage additional

high-entropy keys, CASPER is designed to achieve high detection accuracy while allowing Alice to manage only a low-entropy secret that is easy to remember (e.g., a 2-digit numeric PIN) or retrieve (e.g., the last few digits of Alice's bank account number). Note that, after the initial PMS and device setup, CASPER operates transparently in Alice's view — she can use online services provided by RPs and PMS providers without any additional actions required by CASPER during account login or passkey backup and synchronization. Furthermore, to avoid introducing new usability challenges, CASPER does not rely on users to manage multiple devices for new device setup. We will discuss CASPER's usability in more detail in Section 5.8, including ways to further enhance its user experience.

❸ *No security degradation.* CASPER does not degrade users' account security in any way. In other words, Alice's accounts must be at least as secure as they would be without CASPER deployed. This also implies that, when no PMS passkey breach has occurred, a denial-of-service attacker must not be able to disrupt the availability of RPs' services by abusing CASPER to cause false breach detection.

CASPER overview. Behind the scenes, given Alice's η, CASPER produces a set of detection secrets $W = \{w_1, w_2, \ldots, w_{k+1}\}$ with one of them being the real detection secret represented by w_{i^*}. Then CASPER encrypts Alice's passkey, denoted by s, with a key derived from w_{i^*}. If the encrypted passkey is denoted by $\tilde{s}$, Alice backs up to provider the detection secret set W together with $\tilde{s}$ *instead of* s. If a `caat` has access to $(W, \tilde{s})$ leaked from PMS, they can attempt to decrypt $\tilde{s}$ and recover Alice's passkey by guessing which among W is w_{i^*}. In this case, the attacker may guess a wrong detection secret, derive a wrong key to decrypt $\tilde{s}$, and subsequently obtain a wrong passkey for unauthorized account login attempts. With detection information registered at the RP beforehand, the RP would be able to verify the authentication response with the corresponding decoy verification key, which indicates a potential compromise of users' encrypted passkeys $\tilde{s}$

▷ Step 1. Provider account setup
1. $W \xleftarrow{\$} \mathsf{GenDetectSecrets}(\mathsf{k})$ // *Sec.5.4 discusses* GenDetectSecrets
2. send (aid, W) to provider for backup

▷ Step 2. Authenticator setup
// *If W not saved by the authenticator during step 1*
3. download (aid, W) from provider
// *Get η from user, Sec.5.4 discusses* SelectRealSecret
4. $w_{i^*} \leftarrow \mathsf{SelectRealSecret}(W, \eta)$
5. saves $(\mathsf{aid}, w_{i^*}, W)$ to authenticator

▷ Step 3. Passkey registration
6. $\langle \mathsf{s}_{i^*}, \mathsf{v}_{i^*} \rangle \xleftarrow{\$} \mathsf{KeyGen}(1^\kappa)$ // KeyGen *as specified by FIDO2 standards*
7. $(\tilde{\mathsf{s}}, z) \xleftarrow{\$} \Pi_{\mathsf{EncCred}}(w_{i^*}, \mathsf{s}_{i^*})$
8. $V \leftarrow \Pi_{\mathsf{GenVerifierSet}}(W, \tilde{\mathsf{s}}, z)$
9. sample $V' \subseteq V \setminus \{\mathsf{v}_{i^*}\}$ uniformly randomly s. t. $|V'| = \lceil \alpha \cdot (|V| - 1) \rceil$
10. sends (uid, V', V) to RP and subsequently saved by RP on its credential database
11. sends $(\mathsf{aid}, \mathsf{uid}, \mathsf{sid}, \tilde{\mathsf{s}}, z)$ to provider and backed up by provider for passkey restoration

▷ Step 4. Passkey restoration
12. $\mathsf{s}_{i^*} \leftarrow \Pi_{\mathsf{DecCred}}(w_{i^*}, \tilde{\mathsf{s}}, z)$ // *Get w_{i^*} from authenticator, $\tilde{\mathsf{s}}$,z from provider*
13. saves $(\mathsf{uid}, \mathsf{sid}, \mathsf{s}_{i^*})$ for completing future login requests with sid

Figure 5.3: Passkey Backup and Restoration (BnR) protocol as described in Section 5.5. Notations used are explained briefly in Figure 5.1 and the building blocks in Figure 5.4.

leaked from PMS.

In this chapter, we introduce CASPER in the context of detecting FIDO2 passkeys leaked from PMS. However, the concept of CASPER can also be applied to detect breaches of PMS that support other system-generated authentication credentials, such as randomly generated passwords and long-term seeds for HMAC / time-based one-time passwords (HOTP / TOTP) widely used for 2FA today. For interested readers, we will explain in Appendix C.6 how CASPER can be extended for OTP as another example of CASPER's application.

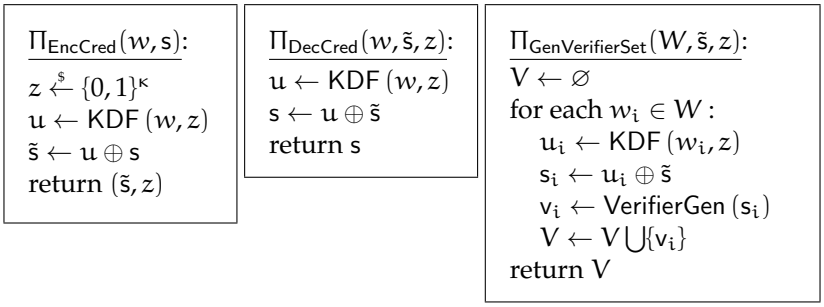


Figure 5.4: Building blocks introduced by CASPER and used by the BnR protocol Figure 5.3. Notations used are explained briefly in Table 5.1.

Backup & restoration protocol

As shown in Figure 5.3, the passkey backup and restoration (BnR) protocol includes the following four steps.

Step 1. Provider account setup. This initialization setup is invoked only once when the user signs up for a new passkey management service (PMS) account. In this step, alongside assigning a unique provider account identifier denoted by aid, the authenticator generates a detection secret set W of size $k+1$ by calling the GenDetectSecrets procedure described earlier in Section 5.4. The authenticator then synchronizes (aid, W) to the PMS provider for backup and saves a local copy optionally.

Step 2. Authenticator setup. Upon successful authentication, PMS typically requires an existing PMS user to set up a new passkey authenticator to enable it for passkey synchronization via PMS. For CASPER, in addition to the usual authenticator setup processes specified by PMS, the authenticator also retrieves (aid, W) from PMS and prompts the user to provide η. The authenticator executes $\mathsf{SelectRealSecret}(W, \eta)$, recovers the real detection secret w_{i^*}, and saves $(\mathsf{aid}, w_{i^*}, W)$. Then the user can start using this authenticator to register new RP accounts (see Step 3) or to log into RP accounts with passkeys restored from PMS passkey backups (see Step 4). Crucially for the security, w_{i^*} is kept private locally from any other parties

including the PMS provider or RPs. The user secret η is never stored at any participating parties.

Step 3. Passkey registration. When the user registers a new account at a RP identified by sid, the authenticator first runs KeyGen to generate a passkey pair (s_{i^*}, v_{i^*}) including the passkey s_{i^*} and its corresponding public verification key v_{i^*}. We can identify this newly registered account by uid and implicitly follow the implementation of KeyGen as defined by FIDO2 standard signature schemes (e.g., ECDSA [133]).

Next the authenticator calls two procedures Π_{EncCred} and $\Pi_{\mathsf{GenVerifierSet}}$ (as shown on the right side of Figure 5.3). Π_{EncCred} returns $\tilde{s}$ — an encrypted version of the user's passkey s_{i^*} for later passkey synchronization and backup via PMS. Given s_{i^*}, $\Pi_{\mathsf{GenVerifierSet}}$ returns a set of verification keys V, the RP for the compromise detection (CD) algorithm we will detail in Section 5.5.

Under the hood to encrypt the s_{i^*} as $\tilde{s}$, the Π_{EncCred} procedure first samples a nonce uniformly at random z and then invokes a key derivation function (KDF) with w_{i^*} and z as its input. KDF returns a *recovery key* u_{i^*} which serves as the encryption key to encrypt s_{i^*} in a "one-time pad" manner. Considering what lies ahead for the CD protocol, however, CASPER needs to ensure that the attacker gets an incorrect but *well-formed* valid passkey s when running Π_{DecCred} with a decoy detection secret $w \in W \setminus \{w_{i^*}\}$. Thus the authenticator should perform additional passkey validity tests. These tests include for all $w \in W \setminus \{w_{i^*}\}$, running $\Pi_{\mathsf{DecCred}}(w, \tilde{s}, z)$ to get s and checking if s is a well-formed passkey. Otherwise, the authenticator should repeat running KeyGen and Π_{EncCred} until all resulting s are well-formed passkeys. This process is efficient because the probability of yielding a well-formed (decoy) private key when $\tilde{s}$ is decrypted with an incorrect w is overwhelmingly high (see Appendix C.4 for more details).

The $\Pi_{\mathsf{GenVerifierSet}}$ procedure takes $(W, \tilde{s}, z)$ as input and generates $k + 1$ verification keys with each corresponding to a detection secret $w_i \in W$.

To do this, for each $w_i \in W$, the authenticator invokes $\mathsf{KDF}(w_i, z)$ to get a recovery key u_i and uses it to decrypt $\tilde{s}$ by running $s_i \leftarrow u_i \oplus \tilde{s}$. Then the authenticator assembles the set of verification keys $V \leftarrow \bigcup_{i=1}^{k+1}\{v_i\}$ by deriving v_i from s_i. Afterward, the authenticator marks a certain fraction (denoted by α) of $V \setminus \{v_{i^*}\}$ uniformly at random as a new subset decoy verification keys, denoted by V', and thus $\alpha = |V'|/(|V|-1)$. Both (V', V) are sent to RP to complete passkey registration[3], and these two sets are used by RP to run the compromise detection algorithm in Section 5.5.

Looking ahead, we refer to V' as the set of *active* decoy verification keys for the RP because, as we will show later in Figure 5.5, an authentication response successfully verified by any of these *active* decoy verification keys will trigger a detection alert. In contrast, other decoy verification keys in $V \setminus (\{v_{i^*}\} \cup V')$ are *inactive* and, together with the real verification key, v_{i^*}, will be viewed as valid verification keys since successful verification by these verification keys will lead to successful logins.

Finally, after receiving the successful registration confirmation from RP, the authenticator synchronizes $(\mathsf{aid}, \mathsf{uid}, \mathsf{sid}, \tilde{s}, z)$ to PMS. Now PMS backs up the $\tilde{s}$ alongside the corresponding $(\mathsf{uid}, \mathsf{sid}, z)$ for the user's PMS account identified by aid. Note that PMS now manages $\tilde{s}$ instead of s_{i^*}. As a result, after a PMS breach, a `caat` cannot accurately derive s_{i^*} from $\tilde{s}$ without η.

Step 4. Passkey restoration. As usual, PMS allows the user to synchronize their passkeys back to a registered authenticator. The registered authenticator first retrieves from PMS a copy of the encrypted passkey $\tilde{s}$ alongside $(\mathsf{uid}, \mathsf{sid}, z)$. It then executes the procedure $\Pi_{\mathsf{DecCred}}(w_{i^*}, \tilde{s}, z)$ to retrieve s from $\tilde{s}$. This procedure, as shown on the right side of Figure 5.3, decrypts $\tilde{s}$ with the recovery key u_{i^*} to obtain s_{i^*}. Finally, the user's authenticator

[3]Here, the authenticator sends V' together with V for better clarity. In practice, instead of the entire V', it suffices for the authenticator to send the RP only a set of identifiers or indices indicating which verification keys among V are in V', which can slightly lower the communication and storage costs.

saves $(\mathsf{uid}, \mathsf{sid}, \mathsf{s}_{i^*})$ so the user can use the restored passkey s_{i^*} to log into their account with the identifier uid at RP identified by sid. For correctness, given a $(\tilde{s}, z)$ pair, observe that $\exists i \in [k+1]$ s.t. $w_i = w_{i^*}$, and thus $u_i = u_{i^*}$. Then we have $s_i = s_{i^*}$ since $s_i = u_i \oplus \tilde{s} = u_{i^*} \oplus s_{i^*} \oplus u_{i^*} = s_{i^*}$.

An alternative design. Instead of backing up a single encrypted passkey $\tilde{s}$ in Step 3, CASPER could alternatively derive and back up all $k+1$ passkeys, i.e., $s_1, \ldots, s_{k+1}$, to PMS. This alternative design could provide the same level of detection effectiveness but would introduce two deployment limitations: (i) PMS storage and communication overhead would increase from $\mathcal{O}(1)$ to $\mathcal{O}(k)$ for each user account; (ii) it would become incompatible with existing PMS designs as existing PMS synchronization and storage implementations are designed to handle one passkey for one credential registration at RPs. We therefore take a different approach by drawing inspiration from honey encryption [134]: instead of directly backing up all possible decoy plaintexts (i.e., passkeys s_i), our design ensures that decrypting the backed up ciphertext (e.g., the encrypted passkey $\tilde{s}$) with incorrect decryption keys yields those decoys. This approach optimizes PMS compatibility while minimizing PMS communication and storage overhead.

Compromise detection algorithm

The passkey compromise detection (CD) algorithm (Figure 5.5) is run by the relying party (RP) upon each login request it receives. It allows the RP to leverage the set of *active* decoy verification keys to detect the `caat`'s account login attempts with corresponding decoy passkeys (see Step 3 in the BnR protocol for producing active decoy verification keys). Next, we describe in detail how the detection is performed for FIDO2.

In FIDO2 (as shown in Figure 5.5), when the RP receives a login request with an account identifier uid, it produces a random challenge and sends it together with other information specified by the WebAuthn pro-

- For a login request for an account uid, the RP checks if uid exists and then returns a challenge to the user.
- Let v, rsp, and γ denote the public verification key, the response produced by the user's authenticator, and the signature generated with the user's passkey, respectively. Upon receiving $(\mathsf{v}, \mathsf{rsp}, \gamma)$ from the user, the RP performs the following tests and actions:
  - retrieves (V, V') corresponding to uid
  - `if` $\mathsf{Verify}(\mathsf{v}, \mathsf{rsp}, \gamma) = \textit{false}$ OR $\mathsf{v} \notin V$, RP rejects this login request.
  - `else if` $\mathsf{v} \in V'$, RP raises a detection alarm.
  - `else` RP accepts this login request.

Figure 5.5: The Compromise Detection (CD) algorithm of CASPER used by the RP to detect passkey compromise.

tocol [15]. Upon receiving the challenge, the authenticator produces an authentication response rsp and generates a signature γ using the private signing key s by running $\gamma \leftarrow \mathsf{Sign}(\mathsf{s}, \mathsf{rsp})$ on the authenticator[4]. Finally, the user device sends $(\mathsf{v}, \mathsf{rsp}, \gamma)$ as a response back to the RP for login.

Once the RP receives the response $(\mathsf{v}, \mathsf{rsp}, \gamma)$, it will perform a set membership test for $\mathsf{v} \overset{?}{\in} V$ and also run $\mathsf{Verify}(\mathsf{v}, \mathsf{rsp}, \gamma)$ to check if the received γ is a valid signature of rsp under the verification key v. If either of these two tests outputs *false*, this login attempt fails. If both tests output *true*, the RP can further check whether v is in the active verification key set V' to determine whether the current login attempt triggers a passkey breach detection or results in a successful login.

[4]The definitions and implementations of Verify and Sign adhere to the FIDO2 standards, which we omit here for simplicity.

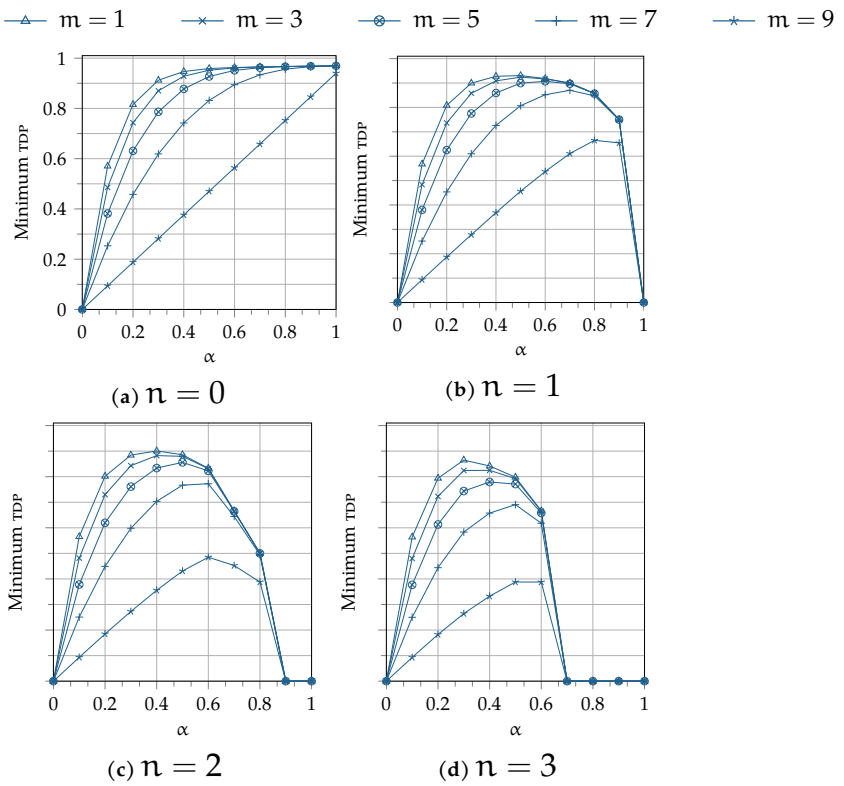


Figure 5.6: Minimum expected true detection probabilities as a function of α with varying m and n, where $\bar{k} = 32$. Here α is the fraction of *active* decoy verifiers V' present in V (i.e. $\alpha \leftarrow \frac{|V'|}{|V|-1}$), n is the number of breached websites `caat` observes, m is the number of websites `caat` wants to compromise.

5.6 Detection Effectiveness

In this section, we show the detection effectiveness of CASPER. First, we show that CASPER does not make it any easier for a `caat` to distinguish the real detection secret from the decoys (Section 5.6). We then estimate true detection probabilities and the overall security benefits CASPER provides through probabilistic model checking (Section 5.6), followed by a false detection analysis (Section 5.6).

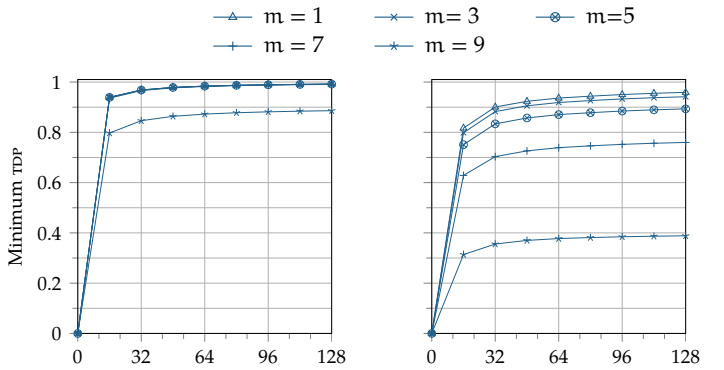
Flatness preservation

Following the informal definition of *flatness* by related work on decoy passwords [128, Sec. 2.1], [132, Sec. 3.1], here we informally define *flatness* as the probability that the `caat` outputs the real detection secret w_{i^*} as its guess *on the first try* given the set of detection secrets W and denote it by $\mathsf{flt}(W, w_{i^*})$. We also say W to be perfectly flat if $\mathsf{flt}(W, w_{i^*}) = \frac{1}{|W|}$ (or equivalently, $\frac{1}{k+1}$). Recall that in Section 5.4, we specified that CASPER relies on $\mathcal{G} = \langle \mathsf{GenDetectSecrets}, \mathsf{SelectRealSecret} \rangle$ that given (η, k) outputs w_{i^*} and W of size $k+1$. We show that CASPER provides a critical security property, *flatness preservation*. We formally define and prove this property in Appendix C.3.

The flatness preservation property of CASPER ensures that given multiple $(\tilde{s}, z)$ pairs from the PMS credential backup compromise for the same user, the `caat` does no better in identifying the real detection secrets, w_{i^*}, from W with the knowledge of multiple $(\tilde{s}, z)$ pairs than without. Briefly, to show this, as further explained in Appendix C.3, we consider a simulator without multiple $(\tilde{s}, z)$ pairs but can by itself generate multiple *simulated* $(\tilde{s}', z')$ pairs by choosing all $\tilde{s}'$ and z' uniformly at random from $\{0,1\}^\kappa$. Since simulated $(\tilde{s}', z')$ pairs are indistinguishable from $(\tilde{s}, z)$, there is no useful information in $(\tilde{s}, z)$ about which among W is w_{i^*} for the `caat` to improve its guessing on w_{i^*}.

True detection and efficacy of CASPER

We provide a comprehensive perspective by estimating both true detection probabilities (TDP) and the overall security efficacy of CASPER. The former provides a simple yet clear picture of how likely CASPER can detect the attack by the `caat` *if the attack occurs*. However, TDP alone falls short of capturing the security efficacy provided by CASPER to users' account security completely in a more realistic setting where the `caat`'s attack



(**a**) $n = 0, \alpha = 0.9$ (**b**) $n = 2, \alpha = 0.4$

Figure 5.7: Minimum true detection probabilities as a function of $\bar{k}$ with varying m, n, and α.

strategy has already been influenced by the deployment of CASPER; it may decide to attack later or stop attacking early. For example, to avoid detection by CASPER, a cautious `caat` may postpone the attacks until a later time to obtain more information about the user. However, PMS may have already discovered its credential backup compromise and asked the user to reset their credentials prior to the `caat` starting such a deliberately delayed attack. In this case, the deployment of CASPER would already add efficacy to the user's account security. To capture this nuanced scenario besides TDP, we measure the efficacy of CASPER by measuring how comparatively well CASPER can reduce the `caat`'s ability to consistently take over user accounts unnoticeably with compromised credentials leaked from PMS.

Modeling detection secrets. Recall that η is the survivable user secret that determines which among W is w_{i^*}. The level of flatness provided by (W, w_{i^*}) and measured by $\mathsf{flt}(W, w_{i^*})$ is in fact difficult to estimate since we do not know (1) the probability distribution of η or (2) how well the `caat`'s strategy together with its knowledge about η can improve its guessing results. Therefore, to have a better generality for our true

detection and efficacy analysis, we follow the similar formal treatments of password guessing from prior work (e.g., [135]). Specifically, instead of providing the caat with the original detection secret set output by $\mathcal{G}$ with flatness $\mathsf{flt}(W, w_{i^*})$, we give an *equivalent* detection secret set $\bar{W}$ of size $\bar{k}$that is *perfectly flat* and provides approximately the same flatness as $\mathsf{flt}(W, w_{i^*})$. This implies the caat can do no better in guessing the real detection secret in $\bar{W}$ with probability $\frac{1}{\bar{k}+1}$, where $\bar{k} = \lfloor \frac{1}{\mathsf{flt}(W, w_{i^*})} - 1 \rfloor$, i.e., the largest integer $\bar{k}$such that $\frac{1}{\bar{k}+1} \leqslant \mathsf{flt}(W, w_{i^*})$.

This abstraction helps us to model the caat's ability to guess the real detection secret in our analysis as a function of $\bar{k}$, avoiding additional (and possibly inaccurate) assumptions on the caat's knowledge about η and its strategies, as well as making our analysis applicable for different types of probability distributions and flatness that η as instantiated in practice may have. To see this consider the case when η that is generated uniformly at random, e.g., a random (numerical) PIN with x digits where $10^x > k + 1$, then (W, w_{i^*}), produced by $\mathcal{G}$ given k and η here, provides perfect flatness, i.e., $\mathsf{flt}(W, w_{i^*}) = \frac{1}{k+1}$ and so $\bar{k} = k$. However, when η is a *user-chosen* password or PIN, the min-entropy [136] of these secrets must be at least $\log_2(\bar{k} + 1)$ bits for a given $\bar{k}$. For example, for $\bar{k} = 32$, if instantiated with a random (numerical) PIN, η should include at least 2 digits such that $10^2 > 32 + 1$; whereas if η is a user-chosen PIN, the min-entropy of η should be at least $\log_2(32 + 1) \approx 5$ bits.

Following Section 5.3, we allow the caat to learn from PMS the detection secret set W sent to PMS during PMS account setup together with $(\tilde{s}, z)$ pairs sent during the RP account registration step by the authenticator. Additionally, the caat observes breaches from n other already *compromised* RPs. These breaches include both the verification key set V and the *active* decoy verification key set V' from each n compromised RPs. Considering that, given a $(\tilde{s}, z)$ pair, for each $v \in V$, there exists a $w \in W$ such that v can verify the authentication response produced by its corresponding s, which

is recoverable by $\Pi_{\mathsf{DecCred}}(w, \tilde{s}, z)$, the caat can rule out exactly $|V'|$ decoy detection secrets in W by observing V' from the first breach snapshot and rule out $\leqslant |V'|$ for each of the rest of the $n-1$ breach snapshots due to that overlapping may exist. Here we use $\bar{W}^{(n)}$ to denote the subset of detection secrets corresponding to verification keys that always fall into $V \setminus V'$ across these n breaches.

The goal of the caat is to compromise the user's accounts at a specified number of *unbreached* RP (denoted by m). Note that the probability of the caat triggering a true alarm when compromising a user's account at an *unbreached* RP is the probability of the event that the caat picks a detection secret from $\bar{W}^{(n)}$, uses it together with $\tilde{s}$ to derive a (decoy) passkey s, and produces an authentication response verifiable by an *active* decoy verification key $v \in V'$ at the unbreached RP (see Figure 5.5).

True detection probability. To capture the caat's best strategy for minimizing the true detection probability (TDP) while achieving its goal, we model the caat as a Markov decision process (MDP). Conceptually an MDP consists of a set of states and potential transitions between them. When the MDP is in a specific state, it can select one of several available actions, which leads to a specific probability distribution over the possible next states. The MDP attacker enters the final state when it triggers a breach alarm raised by CASPER while attempting to compromise the users' accounts at least one of the m *unbreached* RPs. In our experiments, without loss of generality, we consider that the caat is targeting *one user* with a web account at each of the m *unbreached* and n *breached* RP with CASPER deployed. Specifically, using the PRISM model checker [137], we construct an MDP to model a caat who does the following:

- The caat randomly selects a detection secret w from $\bar{W}^{(n)}$, derives a passkey through $\Pi_{\mathsf{DecCred}}(w, \tilde{s}, z)$ and attempts to authenticate at the first RP.
- If the selected detection secret w corresponds to a verification key $v \in V'$

at the current RP, the caat enters the "detection" state and the experiment is over. Otherwise, this RP is considered compromised without detection. If $m > 1$, the caat can choose to either 1) reuse the same detection secret used in the last attack or 2) randomly select a *different* detection secret w', i.e., $w' \xleftarrow{\$} \bar{W}^{(n)} \setminus \{w\}$, and then attack the next RP.

- The caat repeats the above attacks until it either 1) moves to the final "detection" state and triggers an alarm or 2) completes attacking and compromising all m RPs without detection.

The experiment outputs the minimum probability of the best-strategy caat entering the final state over all possible action paths, as reported in Figure 5.6 and Figure 5.7.

Results of TDP experiments. Figure 5.6a-5.6d correspond to four different $n \in \{0, 1, 2, 3\}$ respectively for a fixed $\bar{k} = 32$. When $n = 0$ (Figure 5.6a), a larger α results in a higher probability of the caat triggering a detection alarm because it is more likely that the verification key corresponding to the selected detection secret falls into V'. Also, attacking more RPs exposes the caat to greater risks in triggering a detection alarm *more than one RPs*. However, as n is increased to 1, 2, or 3 as shown in Figure 5.6b, Figure 5.6c, and Figure 5.6d respectively, the detection probabilities for a relatively high α, e.g., 0.9, drop significantly but remain almost unaffected for a relatively low α, 0.1. This is because a higher α leads to a smaller $|\bar{W}^{(n)}|$ given a fixed n, and, as a result, increases the possibility of the caat selecting the real detection secret at the very first attack.

Figure 5.7 shows the minimum true detection probabilities as a function of $\bar{k}$for a given fixed α and the number of RPs n where the caat has observed breaches. One interesting observation here is that when $\bar{k}$reaches a certain level, e.g., 32, increasing $\bar{k}$further can hardly give a significant boost in the minimum probability. We believe this observation provides an informative insight for determining a proper $\bar{k}$that provides a good balance between CASPER's detection accuracy and performance—additional

detection power provided by an unnecessarily larger $\bar{k}$would be limited, but it would negatively affect the overall performance and storage costs of CASPER (see Section 5.7).

Estimating security efficacy. We implement another model-checking experiment to investigate the efficacy of CASPER when the `caat` decides to wait for τ time intervals before starting the attack or stop attacking early after successfully taking over m' ($\leqslant m$) accounts to minimize detection probability. Specifically, once the `caat` have access to a user's compromised credential backups from PMS, the `caat` decides whether to attack immediately (i.e., $\tau = 0$) risking detection with a higher probability (see Figure 5.6, particularly Figure 5.6a) or wait τ time intervals, hoping to observe more RP breaches to lower the detection probability. In that latter case, however, the `caat` has to accept the risk that, over time, the PMS provider will discover the compromise of the user's credential backup itself and notify the user. We also consider that the `caat` may choose to stop the attack early after successfully taking over m' ($\leqslant m$) accounts if the `caat` "believes" compromising accounts at more RPs will lead to a high detection probability.

To understand CASPER's detection efficacy under this realistic scenario, we allow the `caat` to have access to snapshots of a certain number of compromised RPs' persistent storage, which are breached at a Poisson arrival rate with mean λ per time interval. We use a random variable T_{PMS} to represent the time interval soon after which the PMS (provider) identifies by itself the compromise of the user's credential backup *without* CASPER. We define another random variable T'_{PMS}, that is, the time interval soon after which the PMS (provider), if it has failed to detect its breach for the first τ interval, identifies the compromise of the user's credential backup by either itself or CASPER.

Given the public knowledge of λ and T_{PMS}, we build an MDP attacker similar to the one specified in Section 5.6 but with two additional options

for the caat.

- The caat can either choose to immediately start attacking at $\tau = 0$ by attempting to log into each RP (one at a time for simplicity) or to wait for $\tau > 0$ time intervals before starting the attack. At the end of the τ-th interval, if PMS has not identified the compromise by itself in the first τ intervals, the caat can start its attack with persistent storage snapshots of RPs that are breached at a Poisson arrival rate with mean λ.
- After successfully taking over a certain number of accounts, the caat can choose to continue to attack the next or stop its attack early. In the former case, if it triggers detection by CASPER when attacking the next, the experiment ends immediately with $T'_{PMS} = T_{PMS}$. In the latter case, the number of taken-over accounts is fixed, denoted by m' ($\leqslant m$) and we assume that the caat can persistently access m' taken-over accounts until T'_{PMS} when the PMS detects the compromise by itself.

We measure the security efficacy of CASPER by observing how CASPER reduces the expected overall takeover duration of the user's RP accounts and define it as follows.

$$\mathsf{eff} = \frac{\mathbb{E}\left(T_{PMS} \times m\right) - \mathbb{E}\left((T'_{PMS} - \tau) \times m'\right)}{\mathbb{E}\left(T_{PMS} \times m\right)} \tag{5.1}$$

Here $\mathbb{E}\left(T_{PMS} \times m\right)$ is the expected overall takeover duration of the user's m RP accounts until when PMS discovers the credential compromise *without* CASPER. $\mathbb{E}\left((T'_{PMS} - \tau) \times m'\right)$ represents the expected overall takeover duration of m' ($\leqslant m$) accounts that are taken over by the caat with the deployment of CASPER. The MDP attacker's goal is to minimize eff (and maximize $\mathbb{E}\left((T'_{PMS} - \tau) \times m'\right)$) by adopting the best available strategies.

In our experiment, for a reasonable realistic estimate, we see each time interval as one month. Thus assuming a nine-month average compromise

discovery delay as reported in [138], we set T_{PMS} following a normal distribution with mean $\mu = 9$ with a varying standard deviation σ. To interpret that the caat can observe one additional RP breach snapshot after waiting for every 2 and 4 additional months, we set $\lambda = 0.5$ and $\lambda = 0.125$, respectively.

Results of efficacy experiments. To understand the efficacy of CASPER, we explore the effects on eff from (5.1) under different parameterization settings of $(\lambda, \bar{k}, \sigma)$ as reported in Figure 5.8. Figure 5.8a represents the baseline setting where $\lambda = 0.5, \bar{k} = 32,$ and $\sigma = 2$. We modify λ, $\bar{k}$, and σ respectively in Figure 5.8b, Figure 5.8c, and Figure 5.8d from the baseline setting to observe the improvement in efficacy.

We observe a boost in the minimum eff when λ is decreased from 0.5 (5.8a) to 0.125 (5.8b) and $\bar{k}$increased from 32 (5.8a) to 128 (5.8c) as shown in Figure 5.8b and Figure 5.8c respectively for $\alpha \in [0.6, 0.9]$. The main reason behind these boosts is that lower λ (and thus smaller n) or higher $\bar{k}$increase the probability of the caat triggering detection by CASPER, particularly for a relatively large α (see Figure 5.6 and Figure 5.7). So the caat chooses either to bear such an increased risk or to attack later, hoping to observe more breached RPs. This results in shorter account takeover duration and is reflected by the boost of the minimum eff. The effect of modifying σ on the minimum eff is slightly less pronounced as shown in Figure 5.8a ($\sigma = 2$) and Figure 5.8d ($\sigma = 1$) respectively. This is because a smaller σ results in a more tightly centered normal distribution — a stronger detection power of the PMS provider.

One interesting observation is that m does not affect the results for a relatively large α, e.g., when $\alpha > 0.6$ in Figure 5.8a and Figure 5.8d, and $\alpha > 0.8$ in Figure 5.8c. This is because larger α motivate the best-strategy caat to adapt its strategy to increase τ, that is, to wait longer (i.e., with a larger τ) for more breached RP snapshots for a more accurate guess on the real detection secret. With it, the caat can derive real authentication

credentials and log into all $m(=m')$ RP accounts without being detected by CASPER. This, however, shortens the overall account takeover duration from $\mathbb{E}(T_{PMS} \times m)$ to $\mathbb{E}((T'_{PMS} - \tau) \times m)$. As such, eff from (5.1) can be simplified to $\frac{\mathbb{E}(T_{PMS}) - \mathbb{E}(T'_{PMS} - \tau)}{\mathbb{E}(T_{PMS})}$. The elimination of m in this simplified expression explains why m has no effects on eff for these cases.

False detection by a false-alarm attacker

CASPER may raise false alarms if a false-alarm attacker (also known as a denial-of-service attacker in literature, e.g., [110]), without compromising the PMS provider (and thus without knowledge of $\tilde{s}$), successfully derives a passkey s that corresponds to an *active* decoy verification key, that is, $v \in V'$. However, without knowledge of $\tilde{s}$, a guess by a false-alarm attacker can hit one of the *active* decoy verification keys with a probability no greater than $\alpha k/2^{\kappa}$ which is negligible in terms of the credential length κ. For example, for 256-bit private signing keys with k set to 32 and α set to 0.6, the probability of the false-alarm attacker guessing one decoy key that can trigger a false alarm is no greater than $0.6 \times 32/2^{256} < 2^{-251}$ which is negligibly small.

5.7 Experimental Evaluation

Implementation details. We developed a prototype implementation of CASPER [113] in Go language using the open source library virtualwebauthn [139] — which itself is built on top of another WebAuthn library [140]. We first implemented the three building blocks $\Pi_{EncCred}$, $\Pi_{DecCred}$, $\Pi_{GenVerifierSet}$ required during the four steps of the BnR protocol as shown in Figure 5.3. The key derivation function (KDF) used by these three blocks is instantiated using the Password-Based Key Derivation Function (PBKDF2) with

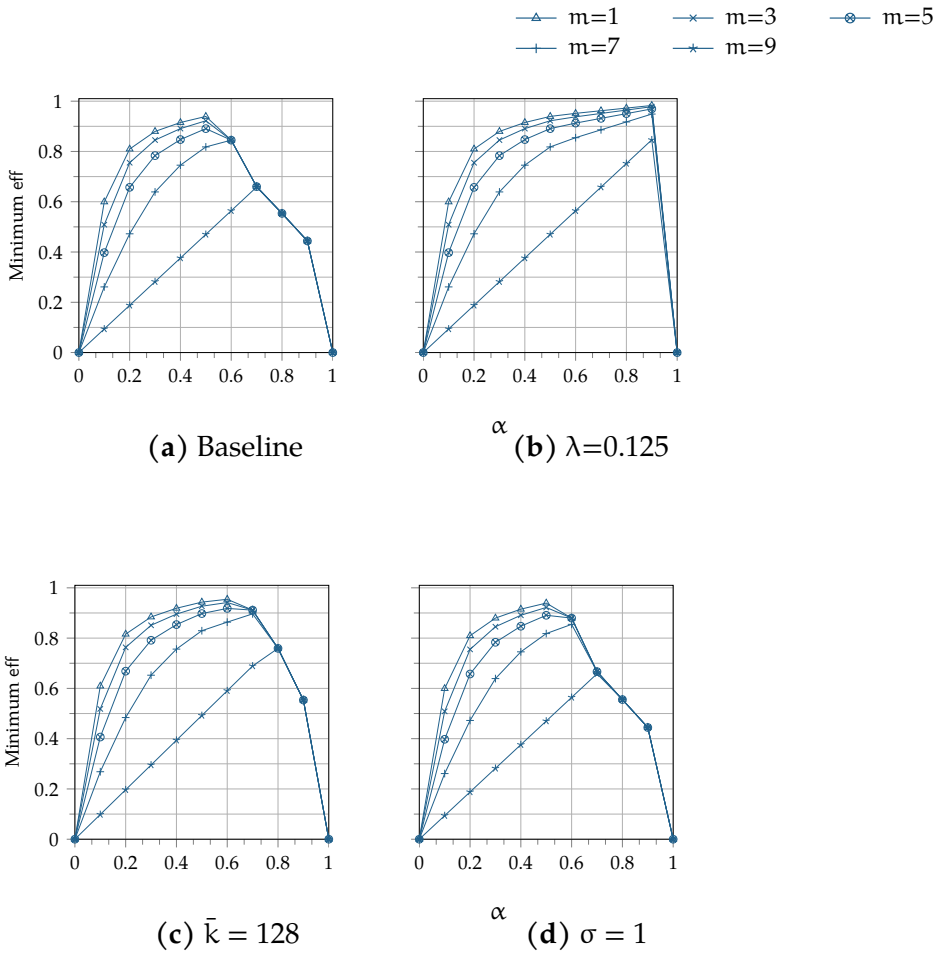


Figure 5.8: eff as a function of α with varying m. Subfigures (5.8b), (5.8c), and (5.8d) show the effects of strengthening security by on parameter, i.e., λ, $\bar{k}$, and σ respectively, from the baseline (5.8a) where $\lambda = 0.5$, $\bar{k}$=32, and $\sigma = 2$.

SHA-256 and 600,000 iterations. For all cryptographic operations, we select the elliptic curve group secp256k1 and set $\kappa = 128$.

To demonstrate the implementation feasibility of CASPER, we instantiated the user secret η with 2-digit system-generated PINs for complete-

ness and set $k = 32$ and $\alpha = 0.6$. We perceive this as a reasonable setting based on our detection analysis presented in Section 5.6, it will have a minimum true detection rate of 0.92 (for $m = n = 1$) and 0.83 (for $m = n = 3$) as observed from Figure 5.6. The virtual authenticator marks $\lfloor \alpha \cdot k \rfloor = \lfloor 0.6 \cdot 32 \rfloor = 19$ of the 32 decoy verification keys as active and sends to the RP all $32 + 1$ verification keys including all decoys and the real, as well as 19 indices indicating which are active decoys.

We implemented the RP and PMS on two separate server nodes also written in Go language. The RP node allows a WebAuthn-supported user account registration and a login end point for the virtual client. For simplicity, instead of using existing APIs PMS from providers such as Apple, Google, or Microsoft, we emulated the same for our PMS node that exposed standard APIs for passkey sync and restoration support. These APIs were then consumed by our virtual authenticator and client. The current prototype implementation did not use TLS to secure the communication channel of the virtual client and authenticator with PMS and RP but is strongly recommended when CASPER is deployed in practice.

Measuring performance overhead. For latency and performance comparisons, we run the PMS (t2.medium) and the RP (t2.micro) on two different AWS EC2 instances running on Ubuntu 20.04 LTS and located in two different regions — US-East and US-West respectively. We run the virtual authenticator and the client on commodity hardware (MacBook Pro M2 with 16 GB of memory). Finally, we registered 10 user accounts at the RP and attempted 25 login attempts on each of the accounts and measured the time delay during account registration and login as experienced by users for CASPER.

When $k = 32$, and $\alpha = 0.6$, our evaluation shows that CASPER introduced an additional delay of 325 ($\pm$ 29) ms to account registration. The majority of the computation time during account registration is taken by the KDF used by Π_{EncCred}, Π_{DecCred}, $\Pi_{\mathsf{GenVerifierSet}}$ procedures of around

224 ($\pm$ 3) ms. Increasing (k, α) would increase the number of invocations of KDF, and thus result in higher delays in user registration and authenticator setup time (step 1 and 2). However, we remark that user registration is an infrequent operation, and will not affect user experience significantly. Importantly, the compromise detection algorithm, specifically the additional membership tests required by CASPER in the algorithm, adds only an average delay of 36 ($\pm$ 8) ms to users' daily login time, which we argue would be hardly noticeable by users.

Benchmarking storage overhead. CASPER adds only minimal storage costs to participating parties. The PMS additionally stores a random value z for each credential entry and a detection secret set W for each user. Each authenticator also needs to store the real detection secret w_{i^*} and a set of detection secrets W. An RP needs to store V and V' for each user account, assuming that there is only one valid credential per account. If z, w, and v are of size 256 bits, and $k = 32$, as a rough estimate for 1 million users with each having 200 RP accounts, the deployment of CASPER would cost the PMS approximately 13.86 Gigabytes and each RP no more than 2.08 Gigabytes in storage only.

5.8 Discussion

We design CASPER to enable RPs to detect attackers' attempts to log into users' accounts using passkeys stolen from a PMS. In Section 5.6, we demonstrate the detection effectiveness of CASPER. In this section, we discuss the usability and deployment considerations of CASPER.

Usability analysis

We propose CASPER as a general detection framework, with the user secret η serving as a flexible component. The usability of CASPER depends on

that of η. Specifically, from an end-user's perspective, CASPER may involve user participation in two operations:[5] (1) registration of η during setup of the first authenticator following PMS *account creation*, and (2) providing the same η to a new authenticator during *authenticator setup* for retrieving passkeys from the PMS.

The user secret η can be a memorized secret selected by the user or assigned by the system, or derived from biometrics or other (long-term) secrets (e.g., bank account numbers or device unlock PINs) of the user. Here we present a brief usability analysis for the case where η is instantiated with 2-digit system-generated PINs [6] to identify the usability caveats and potential mitigation for this instantiation.

PIN-based CASPER: a usability case study. Since CASPER is mostly invisible to users in their daily lives, we focus solely on the following user tasks required by this "PIN-based CASPER": (1) *PIN registration*: During setting up the first device after creating an account with the PMS, CASPER generates a 2-digit random PIN and displays it to the user, instructing them to memorize or note it down. The user is then required to confirm receipt by entering the PIN twice. The last step is repeated if either entry fails to match the PIN displayed. (2) *PIN management*: The user needs to memorize or note down the 2-digit PIN and recall or retrieve it when needed. (3) *PIN entry*: Later, for each subsequent authenticator device setup, the interface will prompt the user to enter the 2-digit PIN twice.

For PIN-based CASPER, the key usability issue is difficulty in memorizing or noting down the PIN, and recalling or retrieving the PIN when setting up subsequent devices / authenticators. This challenge is not particular to CASPER— similar usability challenges also arise in everyday

[5] As we will detail later, certain design options can eliminate the need for user involvement in one or both of the operations. However, for the sake of examining CASPER's usability, we conservatively aim to list all possible user involvement here.

[6] A 2-digit system-generated PIN has approximately $\log_2(100) \approx 6.64$ bits of entropy, whereas η with around 5 bits already provides compelling detection accuracy according to our analysis in Section 5.6.

scenarios where such secrets are commonly used for device unlocking or end-to-end encryption in most cloud-based credential management systems [30]. In these cases, users must effectively manage their PINs to avoid being locked out of devices or losing access to their credential backups. We argue that with PIN-based CASPER, the likelihood of such events is lower, as prior user studies have shown that 2-digit PINs are relatively easier for users to manage [141]. Other usability issues such as PIN input errors during device setup could be avoided by leveraging existing typo-reducing techniques, e.g., [142] or by requesting repetitive PIN entry and confirmation.

While system-generated PINs offer quantifiable security and involve lower deployment complexity, other options to instantiate η, e.g., based on biometrics, can relieve users of memory burdens and address most of the usability issues identified above. For further discussion on alternative options to instantiate η, see Appendix C.5.

Future user studies. To thoroughly evaluate CASPER's usability — including user adoption, perceptions, ease of use, and action accuracy — comprehensive user studies tailored to *specific* η *instantiations and UI designs* are necessary prior to deployment. This chapter does not include such studies, leaving them as future work to better understand CASPER's real-world usability.

Enhancing CASPER's usability

Here we provide suggestions and recommendations to enhance CASPER's overall usability, such as by setting up accurate user perceptions of CASPER, and reducing or eliminating users' efforts additionally required by CASPER.

User perceptions of CASPER. We believe CASPER is appealing to users who: 1) wish to use PMS, 2) have concerns about the security of their PMS passkey storage, as highlighted in a recent user study [21], and 3) prefer

not to adopt less user-friendly mitigations for account security that affects daily login routines (e.g., 2FA / MFA). However, establishing an accurate perception and mental model is a necessary step towards a usable CASPER. To do so, the user interface of CASPER should first clearly detail its goal — to improve users' account security by detecting abuse of passkeys leaked from PMS providers. Second, the interface should clarify that CASPER aims to provide "an additional detection layer" without disrupting users' daily account login routines.

Reducing users' efforts. After η registration, CASPER only requires additional user efforts for η input during authenticator setup, and such instances will be rare. Assuming users set up a new device as often as they replace smartphones (approximately every 40 months [143]), a user would enter η only 32 times over 106 years.

Users' efforts could be even further minimized for the majority of cases when users have at least one registered authenticator available when setting up a new one — for example, when setting up a new smartphone while a previously registered smartphone, tablet, or laptop is available. In such cases, secure device-to-device communication (e.g., similar to Apple's AirDrop) can be used between a new authenticator and a registered one to synchronize w_{i^*} derived from η, eliminating the need for users to input η.

Making CASPER user-invisible. When usability requirements dictate that no additional user tasks should be performed, one may consider extracting η from existing secrets that users already have and use daily, e.g., from PINs / passwords, pattern locks, or biometrics locally for device unlock. In this scenario, it is possible to hide CASPER entirely from users to improve CASPER's usability. However, we caution readers that such a design could introduce additional security implications, particularly if the entropy source is a user-chosen secret without sufficient entropy, which could render CASPER less effective.

Deployment considerations

Deployment requirements. CASPER is by design well-suited for real-world deployment because it is compatible with PMS providers' existing credential synchronization and storage implementations and requires only minimal changes on RPs' side. Implementation-wise, CASPER only requires PMS to additionally store the constant size detection secret set W. For RPs, CASPER requires them to store public verification keys (i.e., V' and V) as specified in the BnR protocol shown in Figure 5.3 and to implement the simple detection algorithm prescribed in Figure 5.5.

However, a potential deployment challenge may arise from the need to modify authenticators to support the BnR protocol and its underlying algorithms. Therefore, to promote the deployment of CASPER, authenticator compatibility would need to be provided by vendors and supported by standardization bodies such as the FIDO Alliance.

Again, instead of a specific instantiation and implementation, we propose CASPER as a general framework to detect centralized PMS. Beyond direct deployment, we hope that our work will inspire future efforts to explore instantiations of CASPER— exploring options such as η instantiations and input methods and alternative methods for generating decoy credentials. These choices should be closely aligned with the real-world requirements for security, usability, and deployment. Moreover, as cryptographic credentials like passkeys and HOTP / TOTP seeds as well as their backup and recovery become increasingly important, we hope our work encourages further research on breach detection for credential backup systems with the goal of improving users' account security.

Risk-based authentication and CASPER. To improve account security, one might seek to compare Risk-Based Authentication (RBA) [144, 145, 146] with CASPER to guide deployment decisions. To help with this process, we provide a detailed comparison of RBA and CASPER to highlight their differences and complementary characteristics.

RBA aims to detect likely unauthorized login attempts by profiling login behaviors and identifying malicious behaviors, without requiring users to manage additional secrets. In contrast, given only a low-entropy η, CASPER provides RPs with a reliable signal indicating whether a user's passkey may have been leaked from PMS. This signal offers deeper insight into the *cause* of the unauthorized account access and allows RP to make more informed decisions, such as requesting users to reset their passkeys to protect their accounts from risks resulting from breaches of PMS. In contrast, RBA aims to flag unauthorized login attempts based on users' login patterns to help RPs to decide if further authentication is needed [145].

RBA relies on login information, e.g., IP addresses, user-agent strings, and user login patterns, which is *independent* of the deployed authentication schemes. This makes RBA compatible with a wide range of authentication schemes, although it is mainly deployed to complement password-based authentication today. CASPER, on the other hand, is a detection framework specifically designed to identify PMS breaches that leak synchronized cryptographic credentials like passkeys and HOTP / TOTP seeds.

In terms of security assumptions, the effectiveness of RBA depends on whether the login information used to identify unauthorized login attempts could be stolen and spoofed. However, existing RBA mostly relies on non-private login information like geolocations and user-agent strings [144], which can be easily obtained and spoofed by a sophisticated attacker to effectively reduce RBA's accuracy [74]. In contrast, CASPER remains effective against the same sophisticated attacker. As shown in Section 5.6, CASPER achieves high detection accuracy even when the attacker is allowed to breach multiple RPs to gather information for attacking other RPs.

Social engineering attacks against CASPER. Attackers could use social engineering techniques like phishing [147, 148] or pretexting [148] to steal

the user secret η. User secrets required by existing PMS such as iCloud and Google backup are also susceptible to similar social engineering attacks. We note that the leak of a user's η only renders the detection ineffective — as if CASPER had never been enabled for this user — but would not degrade the security of their passkeys or RP accounts.

Detection notifications. RP may choose to notify the victim user of the potential compromise of their passkeys from the provider. We discuss two ways the RP could notify the victim user: actively or passively. For active notification, RP can promptly send detection notifications to users via established communication channels (e.g., email, app notifications, SMS). When all communication channels between the user and the RP are unavailable, the RP could also resort to a passive notification such as notifying the user on its webpage or application when they log into their account next time. With such notification, the user can decide to reset their passkeys, better secure their provider account, or even consider switching to a different provider.[7]

Handling spoofed detection alerts. In this chapter, we make no effort to address the possibility that participating RPs might withhold detection alerts from users when detection happens, given that these RPs could equally well do so by simply not participating. However, we have to take into consideration that participating RPs may attempt to send spoofed detection notifications to users when detection does not happen (e.g., just to wrong an unbreached PMS provider). CASPER can provide a strong guarantee for identifying such spoofed notifications by additionally requiring the RP to return to the user the authentication response $(v', \mathsf{rsp}', \gamma')$ that triggers the breach alert as a "proof of detection".

[7] We note that recovering from such an identified compromise, as required by other account recovery needs as they exist today [149], requires users to register at the RP for a secure secondary authentication method (or sometimes termed backup/fallback authentication) in advance to ensure the user's legitimacy for resetting credentials.

Specifically, for the FIDO2 case, the user (device), after receiving (v', rsp', γ') as the proof of detection, first performs $\mathsf{Verify}(v', rsp', \gamma')$ to check that the detection proof is well-formed. Given that γ' is a valid signature of rsp' under v', the user device then runs $\Pi_{\mathsf{GenVerifierSet}}$ to regenerate V, and checks if $v' = v_{i^*}$ OR $v' \notin V$. If this check is true, the user device can confirm this is a spoofed notification because only a (rsp', γ') pair verifiable by one of the verifiers in $V \setminus \{v_{i^*}\}$ can trigger a detection alarm. Informally, without the knowledge of the user's private signing key s_{i^*} or a `caat`'s presence (i.e., compromise of users' credential backup at provider), it would be difficult for an RP to forge a valid detection proof if the underlying digital signature scheme is secure.

5.9 Conclusion

In this chapter, we present CASPER, the first framework to detect the abuse of users' passkeys leaked from passkey management services (PMS). CASPER can be seamlessly integrated into the existing FIDO2 authentication protocols without disrupting users' daily account login routines. Additionally, CASPER is compatible with existing PMS implementations and introduces minimum storage and computation overhead to participating parties. We demonstrate that CASPER provides compelling detection effectiveness, even against attackers who exploit information from website breaches to optimize their strategies to avoid detection. We believe that the widespread deployment of CASPER will enhance users' account security particularly in scenarios where a PMS provider fails to protect users' passkey storage from compromise.

6 CONCLUSION AND FUTURE WORK

The dynamic nature of online user authentication ensures that the ongoing battle between attackers and defenders will persist. As new authentication mechanisms emerge, so will novel vulnerabilities and attack vectors, necessitating continuous advancements in detection and defense strategies. In this chapter, we highlight several promising directions for future research and development.

Robustness of Detection Mechanisms

The deployment of detection mechanisms is not a one-time solution but an evolving process. As demonstrated in this thesis, attackers continuously adapt to bypass security measures, requiring detection strategies to be resilient, adaptable, and capable of countering sophisticated attack patterns. Future work should focus on enhancing the robustness of these mechanisms against adversarial tactics and evasive techniques.

Shifts in Attacker Behavior

While the industry is increasingly adopting passkeys, passwords are unlikely to be eliminated entirely, particularly in account recovery mechanisms. In the near future, passkeys may become the primary authentication method, but passwords will likely remain integral to recovery processes. This shift will incentivize attackers to target account recovery endpoints rather than primary authentication mechanisms. Similarly, as passkey cloud storage gains prominence, it will become an attractive target for exploitation, necessitating stronger security measures to protect stored credentials.

Emerging Classes of Attacks and Defenses

The evolution of security threats is driven by advancements in technology, including generative AI (GenAI). Attackers are likely to leverage AI-driven techniques to develop more effective phishing, guessing, and evasion strategies. Consequently, future defenses must incorporate AI-driven detection and mitigation mechanisms to stay ahead of emerging threats.

Security of Implementations

Vulnerabilities in password storage implementation have historically led to catastrophic security breaches. Compared to passwords, passkeys involve more complex cryptographic and software components, increasing the likelihood of implementation flaws. As passkeys gain widespread adoption, it is critical to assess and mitigate potential security risks stemming from insecure implementations, ensuring that the transition to passwordless authentication does not introduce new attack surfaces.

Addressing these challenges will be crucial in securing the next generation of authentication systems, requiring ongoing research, collaboration, and innovation in the field of security and privacy.

A SUPPLEMENTARY MATERIAL: CHAPTER 3

A.1 Measurements Taken

We show all data stored in the ephemeral and persistent databases in Table A.1. Note that the raw password is only stored in encrypted format for 24 hours in the ephemeral database.

The symbol $\times$ indicates that a field was stored in plain text and $\bigotimes$ indicates it was encrypted before storing in the indicated level of storage.

A.2 Filtering Out Attacks

We excluded high volume attacks before reporting summary statistics to capture an accurate description of regular user behavior. The remaining harder-to-detect adversarial behavior appears to be an insignificant fraction of logins. We were given access to compromise account reports from the instrumentation time period (as described in Section 3.3. We found that an average of 190 compromised accounts were reported every month. Overall, less than 1% of the total user population in the monitoring period were compromised, less than 1% of all logins were to accounts that were compromised at some point in the measurement period, and 2% of IPs were associated with those requests.

The compromise report database logged compromise usernames, but not specific requests or IP addresses corresponding to the user's compromise; thus it was difficult to exclude all attacks without filtering out an even larger portion of benign behavior. Excluding all login requests to usernames that were compromised at any point in our instrumentation period did not significantly change the summary statistics we reported. However, filtering out the high-volume attacks, as we did in the chapter, did change some statistics.

To concretize this, one statistic that we believe is more sensitive to whether attack traffic is included in measurements is the fraction of requests using a breached password. With the high-volume attacks included in the calculation, this fraction was 4.73%. This percentage decreased to 2.36% when we filtered out high volume attacks, but only further decreased to 2.34% when we filtered out all requests associated with a compromised username.

Characterize benign behavior in the presence of attacks is a fundamental challenge, given the lack of ground truth. Future work improving the identification of adversial behavior can confirm our characterization of benign user behavior.

A.3 Login Statistics

We show some additional statistics about the login requests recorded by Gossamer. Table A.3 shows some additional statistics we reported earlier in Table 3.4. Figure A.2 shows the distribution of operation systems as parsed from the user agents of the requests at both universities; Table A.4 shows the top 10 most common user agents we saw at both schools.

Field	Eph.	Pers.
Basic statistics		
username	⊗	⊗
password	⊗	
IP address	×	×
receipt timestamp at the login server and at Gossamer	×	×
receipt timestamp at Gossamer	×	×
HTTP headers	×	×
result (success or failure), result code	×	×
zxcvbn score (bucketized to 0, 1)		×
password was malformed		×
Credential stuffing measurements		
username appeared in the breach data		×
password appeared in the breach data		×
username-password pair appeared in the breach data		×
breach source of the username, password, or pair		×
Credential spraying & dictionary-based guessing measurements		
password appeared in		
– top {10, 100, 1,000} most common breached pws		×
– top {2,000, 5,000} hashcat-generated pws		×
– top {2,000, 5,000} RockYou pws		×
was password frequently submitted today?		×
was username frequently submitted today?		×
Credential tweaking measurements		
PPSM [60] strength of password		×
guess rank due to credential tweaking attack [60]		×
edit dist. $\leqslant 2$ of pw from		
– other submissions for same username		×
– other submissions for same IP		×

Table A.1: Measurements we log in ephemeral (**Eph.**) and persistent (**Pers.**) storage.

OS	U1	U2
Windows	36.13%	23.96%
Mac OS X	29.47%	19.58%
iOS	26.20%	43.47%
Android	5.04%	3.18%
Linux	2.43%	0.31%
Other	0.73%	8.96%

Table A.2: The distribution of operating systems (OS) as detected in the user-agent of all the requests (after removing requests containing empty user-agent string at U2).

	Univ. 1	Univ. 2
Submitted password statistics		
% req. w/ password in breach†	2.71%	0.10%
% req. w/ username in breach†	5.31%	3.08%
% req. w/ user-pwd pair in breach†	0.07%	0.01%
% failed req.		
– containing a typo	29.67%	12.04%
– containing a typo (with edit dist msmt)	62.39%	58.37%
– from mobile device containing a typo	38.63%	38.36%
– from mobile device containing a typo (with edit dist msmt)	72.69%	81.87%
% pwds tweaked	0.92%	0.34%
% pwds w/ zxcvbn score of 0	0.06%	0.40%
% pwds in top 5k hashcat	$< 0.01\%$	0.06%
% pwds in top 5k rockyou	0.02%	0.17%
% pwds in top 1k breach comp	0.01%	0.10%
Session Statistics (with a 360s threshold)		
Avg. session size	2.25	2.21
99th percentile session size	10	6
% abandoned sessions	5.47%	1.96%
% sessions with at least two attempts	22.24%	38.22%
% mobile sessions	41.32%	35.45%
% sessions with a typo	2.64%	0.85%
% mobile sessions with a typo	0.01%	0.20%
Avg. num sessions per user per day	1.74	9.23
User Statistics		
# of unique usernames seen	196,424	309,801
# of valid users	177,286	169,774
# of active users	130,695	110,476
% valid users w/ weak passwords	0.03%	0.06%
% valid users w/ username in breach†	5.79%	3.27%
% valid users w/ passwords in breach†	2.92%	9.34%
% valid users w/ user-pw pair in breach†	0.01%	0.15%
% valid users w/ tweaked password	1.22%	0.66%
% valid users w/ no failed attempts	33.21%	58.02%
% valid users who may be using pw managers	24.76%	27.34%
Avg. fails before a success	1.18	1.19
Avg. devices per user per day	1.51	1.91
Avg. devices per user	14.51	14.97
Avg. IPs per user	8.70	10.56
Avg. successful IPs per user	10.65	17.63
Avg. user agents per user	6.15	3.99
Avg. unique passwords per user	1.96	9.59
Avg. attempts per unique IP per user	5.86	5.21
Login Statistics		
Avg. Login requests per day	49,302	246,274
Avg. # of submitted usernames per day	24,822	61,798
% of requests succeeded	94.99%	92.35%
% of requests with null user agent	0.48%	34.31%
# of requests per day per user		
– Average	1.99	2.05
– median	1	2
– 99^{th}-percentile	7	12
% of requests from mobile device	31.00%	35.57%
% of failed requests from mobile device	24.90%	11.96%

† Statistics related to breach data were calculated for data beginning Jan 27 '21 after we added more breach data to the instrumentation.

Table A.3: Summary statistics of login requests recorded by Gossamer at U1 (from Dec. '20 to July '21) and U2 (from Dec '20 to Mar '21).

User agent	% req.
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36	1.54%
Mozilla/5.0 (iPhone; CPU iPhone OS 14_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0.1 Mobile/15E148 Safari/604.1	0.99%
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36	0.36%
Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:78.0) Gecko/20100101 Firefox/78.0	0.23%
Mozilla/5.0 (Unknown; Linux x86_64) AppleWebKit/534.34 (KHTML, like Gecko) PingdomTMS/0.8.5 Safari/534.34	0.22%
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36	0.21%
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_6) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0.1 Safari/605.1.15	0.15%
Mozilla/5.0 (Macintosh; Intel Mac OS X 11_1_0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36	0.14%
Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:84.0) Gecko/20100101 Firefox/84.0	0.13%
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36 Edg/87.0.664.66	0.13%

User agent	% req.
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36	3.15%
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.141 Safari/537.36	1.56%
Mozilla/5.0 (iPhone; CPU iPhone OS 14_2 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0.1 Mobile/15E148 Safari/604.1	1.54%
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.150 Safari/537.36	1.32%
Mozilla/5.0 (iPhone; CPU iPhone OS 14_3 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0.2 Mobile/15E148 Safari/604.1	1.16%
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.104 Safari/537.36	1.12%
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86.0.4240.198 Safari/537.36	0.88%
Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.182 Safari/537.36	0.76%
Mozilla/5.0 (iPhone; CPU iPhone OS 14_4 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/14.0.3 Mobile/15E148 Safari/604.1	0.70%
Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/87.0.4280.88 Safari/537.36	0.62%

Table A.4: Top 10 most common user agents at U1 (**top**) and at U2 (**bottom**)

B.1 Reasons for Compromise Reports

As described in Section 4.3, we received the logs of compromised account reports from both universities. We show the breakdown of the reasons for compromise as recorded at U1 in Table B.1. We that found 69% of accounts were reported as compromised due to large-scale automated attacks (referred to as "Bulk credential testing" at U1). For compromised accounts reported within one hour of the time of attack, 46% were classified as "Bulk credential testing", 17% as self-reported password compromise, and the remaining 37% were split between the other 10 reasons appearing in the dataset. For the six compromised accounts that took longer than 300 hours (12.5 days) to report, all were reported as "Simultaneous use from different locales."

At U2, the recorded reasons for compromise are very coarse: 99% are reported simply as "compromised accounts". Three accounts were disabled based on requests from the human resources department (former employee), and two were disabled as the user is "deceased". No username is reported more than once for different reasons.

B.2 Using DAS to Detect Attacks

The DAS algorithm [31] has been used successfully in other contexts, such as detecting spearphishing attacks. Applied to our context, the DAS algorithm takes a set $\mathcal{L}_1, \mathcal{L}_2, \ldots$ and orders the sets as follows. For some configured subset of numerical features, we first associate an ordering operator over possible feature values (e.g., higher failure rate FF is more suspicious). Then we associate to $\mathcal{L}_i$ a score that is equal to the number of other sets $\mathcal{L}_j$ $(i \neq j)$ such that $\mathcal{L}_i$'s features are all strictly more suspicious

Reason	Count	Percent
Bulk credential testing	1255	69.3%
Simultaneous use from different locales	438	24.2%
Self-reported password compromise	45	2.5%
Blocked for spamming via Office 365	23	1.3%
Password misused	20	1.1%
Spamming via Office 365 / Gmail	11	0.6%
Other (incl. reports from Duo fraud, third-party, etc.)	20	1.1%

Table B.1: The reasons for compromise as noted at U1 and the number of instances of such compromise. Users reported multiple times are counted as distinct instances.

than $\mathcal{L}_j$'s features. We can then obtain a partial ordering over the set of $\mathcal{L}$ sets based on these scores.

In finding a configuration of features for DAS, we optimized for the fraction of the top 50 $\mathcal{L}$ sets as ordered by DAS that were associated with a compromised username, as reported by the security engineers. In doing so, we found that using two simple volumetric features—the number of requests submitted (NR) and the number of unique users contacted (NU)—yielded one of the highest fractions of $\mathcal{L}$ sets associated with a compromised username. In fact, computing DAS with the features NU and NR yielded 41 out of the top 50 $\mathcal{L}$ sets and 87 out of the top 100 $\mathcal{L}$ sets associated with a compromised username. Adding as a feature the number of consecutive days an IP has been active (CD) increased it to 50 out of the top 50 and 98 out of the top 100. Thus we hypothesize that these three features are most correlated with the current mechanisms used by the university IT offices for detecting attacks that cause compromise reports.

DAS with NR, NU, and CD may only be useful for discovering attacks already being caught by existing countermeasures, so we explore extending to further features that the IT offices may not be considering. For example, features such as the fraction of passwords in a breach (FPIB), the fraction of tweaked passwords (FTP), and average unique passwords per user (AUP) may all help in detecting attacks. We tried running DAS on

a configuration with a much richer feature set: NR, NU, FPIB, FCIB, FTP, AUP, and FF. Despite manually flagging all 50 as probable attacks, only 19 were flagged as attacks in the compromise database. This indicates that the IT offices may be missing many attacks that could be found with a richer combination of features, and it also suggests that the compromise reports may not be a good ground truth.

Our experiments with DAS show that it has promise for discovering attacks. While naively using it with just volumetric features seems to miss more subtle attack behaviors, when used with richer Gossamer-enabled features DAS can even discover (successful) attacks that are not being caught by existing countermeasures. This suggests that future deployments may want to consider using DAS-style approaches for remote guessing attack detection, similar to its original use with spearphishing. However, from the perspective of our goal of better characterizing attack campaigns, DAS has various limitations. In particular, it cannot group IP addresses into attack campaigns, and distributed attacks that use multiple IPs will be treated as separate attacks.

Thus we consider clustering as an unsupervised approach to grouping suspicious IP addresses into attack campaigns.

B.3 Additional Clustering Results

Clustering Quality. Before settling on agglomerative clustering for grouping $\mathcal{L}$ sets into potential campaigns, we tried a number of other clustering techniques. We considered the K-means++, DBSCAN, HDBSCAN, and agglomerative clustering techniques in this study on the same set of features from Table 4.2. Since K-means++ cannot work with a custom similarity model, we applied principal component analysis (PCA) on all feature values of $\mathcal{L}$ sets and projected them in a two dimensional space to run K-means++ clustering. We also tried projecting to more than two dimen-

Clustering algorithm	Silhouette score ↑ U1	U2
PCA and K-Means++	−0.13	−0.09
DBSCAN	−0.39	−0.30
HDBSCAN	−0.12	−0.08
Agglomerative	+0.19	+0.17

Table B.2: Silhouette scores of different clustering models.

sions but did not observe any noticeable effect on the clustering quality. To judge the quality of the clustering techniques, we used the silhouette score [100], which computes the normalized difference between the average inter-class and intra-class distances. We did a grid search over all hyperparameters for each of the clustering techniques and reported the best silhouette score in Figure B.2.

All clustering methods except agglomerative clustering received a negative silhouette score, signifying poor quality clusters. We hypothesize that the poor performance is because our similarity model is not a metric similar to Euclidean distance, which is essential for K-means++ to produce meaningful clusters. DBSCAN performed well when the clusters were of the same density—that is, when $\mathcal{L}$ sets belonging to the same cluster are uniformly distributed inside a cluster. However, in our use case, it is common to have attack campaigns of different densities. While HDBSCAN [150] is specifically designed to handle non-uniform clusters and produced relatively better clusters for a number of attack campaigns, it still received a lower silhouette score than agglomerative clustering.

Sensitivity of clustering threshold. To analyze how sensitive our clustering results are for different thresholds, we run two experiments by changing (a) the filtering threshold which is applied to $\mathcal{L}$ sets and (b) the distance threshold that dictates whether two clusters would be merged or not. Our findings show that the clustering results are sensitive to changes in the distance threshold but remain relatively the same for changes to

filtering thresholds.

For the first experiment, we change the filtering threshold to a less aggressive 70^{th} percentile instead of the 80^{th} percentile as we did originally, while keeping the distance threshold the same as before at 0.51. After manual analysis, we find no noticeable change in the likely malicious clusters in comparison to Araña's clustering results. This indicates that clustering results may not be sensitive to changes in the filtering thresholds.

In the second experiment, in addition to the filtering thresholds, we also change the distance threshold by applying a knee locator method on $\mathcal{L}$ sets filtered by the 70^{th} percentile. This gives a distance threshold of 0.41, which is lower than the original distance threshold of 0.51. Although the majority of the resulting clusters remain the same, we observe a few noticeable changes in the clusters as we describe below.

We sampled the top 20 (16 untargeted and 4 targeted) likely malicious clusters using the same sampling criteria used in Araña. The first 16 clusters had $\mathsf{NR} \geqslant 5,000 \vee \mathsf{NU} \geqslant 5,000$, exhibiting high volume, untargeted behavior. Out of these 16, we discover that 9 clusters were exactly the same as the ones found in Araña's clustering results. However, we identify five new untargeted clusters which we did not see in Araña's clustering result. Our manual analysis confirms one of them to be a malicious attack campaign, one to be clearly benign behavior, and the other three to contain a mix of benign and malicious behavior. Furthermore, we observe that two attack campaigns which were previously split into five clusters by Araña, are more accurately represented by two distinct clusters with this new distance threshold. The remaining 4 clusters with $\mathsf{AUP} \geqslant 25$ exhibited targeted behavior and were identical to those found by Araña.

Lastly, we notice that, with the new relaxed distance threshold, two targeted attack campaigns detected by Araña are completely missed. This is because lowering the distance threshold introduced spurious $\mathcal{L}$ sets that did not exhibit the same targeted behavior as the other malicious $\mathcal{L}$ sets

showing clear targeted behavior. Thus, the AUP value of the mixed cluster was reduced below our selection threshold 25.

In conclusion, we find that Araña's clustering results are particularly sensitve to changes to the distance thresholds. While lowering the distance theshold can allow for the discovery of more potential attack campaigns, it also increases the chance of mixing benign and malicious $\mathcal{L}$ sets in the same cluster. To prioritize precision and a low false positive rate, we choose the higher percentile filtering thresholds which result in a high distance threshold, minimizing the chance of benign and malicious $\mathcal{L}$ sets appearing in the same cluster.

Clustering without password-based features. To understand the importance of recording users' password-based information, we reran Araña without the six password based features at U2. We found that without these features, we were still able detect 92% (1,570) of 1,709 malicious $\mathcal{L}$ sets and 80% (16) of 20 attack clusters discovered by Araña as shown in Table 4.5. However, it missed one multi-day credential stuffing attack from a single IP address (cluster #15) and three targeted attacks (cluster #22, #23, #29). We hypothesize that since all $\mathcal{L}$ sets in attack cluster #15 have similar values across five password guessability features and all $\mathcal{L}$ sets in attack cluster #22, #23, #29 have similar feature values for AUP, Araña placed them in the same respective attack clusters predicting they are originating from the same attackers even if they were spread across multiple days or targeting same users via multiple IP addresses.

Additionally, clustering without password-based features flagged 46 new $\mathcal{L}$ sets. We manually analyzed these 46 $\mathcal{L}$ sets. We suspect that at least four $\mathcal{L}$ sets are malformed clients, since they submitted the same password against a single user. Moreover, we found that $\mathcal{L}$ sets within the same cluster have different values for password-based features. For example, eight $\mathcal{L}$ sets formed a new cluster where one IP address had $\mathsf{FPIB} = 0$, and the other seven $\mathcal{L}$ sets have $\mathsf{FPIB} > 0.55$ with an average

Attack # [11]	# $\mathcal{L}$ sets flagged [11]	Clusters	# $\mathcal{L}$ sets flagged by FCA Flagged	Not Flagged
#1 (at U1)	7	1,6	6	1
#2 (at U1)	1	2	1	0
#3 (at U2)	12	10	12	0

Table B.3: Confusion matrices of the number of $\mathcal{L}$ sets corresponding to the three attack campaigns found in prior work [11] using their manual approach versus our FCA approach.

of 0.66. For the remaining 34 $\mathcal{L}$ sets, it is difficult for us to judge them as either fully malicious or benign, since the password-based features varied within a cluster. Thus we believe password-based features are important for grouping attack traffic into attack campaigns, as well as manually investigating a clustering being malicious or not.

B.4 Comparing Araña to prior work [11]

In Section 4.6, we discuss the three attack campaigns manually identified in prior work [11]. Here we compare in more detail how their attack identification compares to the corresponding clusters found using Araña. In Figure B.3, we present three confusion matrices—one for each attack. These show how many $\mathcal{L}$ sets were labeled as part of the attack, or not, for both methods. For two attacks, agreement was perfect (unsurprising for attack #2 which only emanated from a single IP on a single day), and for attack #1 Araña missed just one $\mathcal{L}$ set. We believe that this false negative arose because that $\mathcal{L}$ set set fell on the next day (after midnight) compared to the prior $\mathcal{L}$ sets, suggesting that there can be some noise introduced by our 24-hour cut-offs for login sets. Even so, an analyst using Araña could, in this case, easily detect the false negative manually since the IP address is the same.

Country	# of requests
United States	65,474
Germany	6,170
Morocco	3,467
Canada	152
Brazil	112

Country	# of requests
United States	234,515
Russia	19,003
Ireland	13,130
Netherlands	5,678
Canada	1,571

Table B.4: The five most common countries at U1 (**left**) and U2 (**right**) by number of attack requests.

B.5 Geographical Source of Attacks

We use the ISP of a request to determine the country or countries of origin. Since a campaign often contains multiple IPs, there may be more than one country of origin. Table B.4 shows the most common countries from which attacks originated. At U1 and U2, the vast majority of malicious IP addresses originated from within the United States.

B.6 Per-day Araña and its robustness

In the clustering approach used for Araña discussed in Chapter 4, $\mathcal{L}$ sets were clustered into attacks campaigns using a set of volumetric, client-level, and password-based features. However, their clustering approach relied on seeing the whole time period of data before applying the clustering, which does not allow an analyst to detect attacks in a timely fashion. Simply applying the Araña approach on a day's worth of login data yields low precision. Instead, we use DAS-based scoring (discussed in Section 2.3) in conjunction with Araña clustering, allowing it to detect attacks in a timely and effective manner.

Two observations motivated us to adopt DAS-based approaches. First, as Ho et al. argue, supervised ML approaches do not handle well the heavy imbalance between normal and anomaly events. While Ho et al. investigated this in the context of spear phishing, we conjecture that similar

dynamics arise in malicious login detection. For example, in Gossamer logs, only 18% of logins at U1 and 11% at U2 are malicious. Secondly, in most real-world login datasets, there is no way to describe benign behavior as a set of patterns or distributions, which is essential for standard anomaly detection approaches [151, 152]. Therefore, we use DAS which can overcome these limitations.

To do this, we first run Araña on the login requests in a given time period to obtain a set of clusters. We then calculate the combined feature values of the resulting clusters (by taking an average or sum of the values), and we compute the DAS score for each cluster w.r.t. each other cluster using each set of sub-detector features separately. As before, any positive DAS score indicates that a cluster is malicious, and if a cluster is flagged malicious, we consider all $\mathcal{L}$ sets (and therefore all requests) within that cluster as malicious.

Performance of Per-day Araña

Araña per-day approach was able to rediscover 28 out of the 29 attack clusters presented in Table 4.5 in a timely fashion. For the missed attack cluster #12, Araña per day could discover one (out of three $\mathcal{L}$ sets) discovered as malicious by the earlier approach. More importantedly Araña per-day was able to flag these malicious attack clusters without introducing too many false positives. For example, the five clusters that were considered malicious by Araña and not by its non-timely counterpert — four of these were manually analyzed to be clearly malicious.

Robustness of Per-day Araña

A major concern when deploying an attack detection approach has been how robust such an approach would be against adversaries that modify their attack strategy based on the defense to evade detection. An attacker

can add dummy requests, slow the frequency at which they submit their login requests, or use multiple IP address to distribute the requests in an attempt to hide the attack signature.

Admittedly given sufficient resources — time, IP addresses, and internet bandwidth — there are always straightforward evasion attacks for any of the defense mechanisms we have considered. For example, given sufficient time, an attacker could send one request per day. Given sufficiently many IP addresses, the attacker could send one request per IP address, which no detection mechanism can currently differentiate from benign login traffic. Thus, a good detection mechanism should ensure that evading detection requires a large amount of resources (in terms of unique IP addresses and time). Conversely, an optimal evasion attacker is one who minimizes the resources required for a maximal evasion success rate. Since solving this minimax optimization problem is challenging, we adopt a practical approach as follows.

Practically, we consider three types of resources that an attacker can use specifically for evading detection: additional IP addresses (θ_{ip}), additional chaff requests (θ_r), and additional time (θ_t).

Evasion strategies. Based on the way resources are used by the evasive attacker, we devise different evasive strategies to test the robustness of the three DAS-based detection approaches. These evasive strategies can be, launched easily with existing attack scripts or tools like OpenBullet2 [153] and Sentry MBA [103] and they are indicative of what an attacker may try in evading detection mechanisms.

Broadly, these evasive strategies can be categorized into two classes: *introducing chaff* which distracts the detection algorithms from the actual attack traffic and *blending* the attack traffic into the benign background traffic. This list is not exhaustive; an attacker may find other ways to craft evasive attacks.

(1) Introducing chaff $\mathcal{L}$ sets. Attackers can introduce chaff to distract the

detector from the actual attack by sending malicious-appearing, non-attack logins from their set of IP addresses. By introducing chaff, the attacker could create new $\mathcal{L}$ sets that score higher than the existing $\mathcal{L}$ sets in the comparison set so that the chaff $\mathcal{L}$ sets entirely populate the comparison set; these chaff $\mathcal{L}$ sets would be designed so that the attack $\mathcal{L}$ never has a relative DAS score higher than these chaff $\mathcal{L}$ sets.

(2) Blending in using sacrificial logins. Attackers can also attempt to blend their malicious login requests into the background benign login requests. To do this, the attacker can add additional spurious login requests with their attack, which we call *sacrificial requests*, as they aim to reduce (or hide) the suspiciousness of malicious $\mathcal{L}$ sets by making them appear more like the surrounding benign traffic.

For example, consider an attack $\mathcal{L}$ originally containing 100 requests. If 40 requests from this $\mathcal{L}$ contain a breached password, the fraction of breached passwords (FPIB) feature would be 0.4. Now if the attacker sends 100 additional sacrificial requests containing a password not present in breach data, that attack $\mathcal{L}$ would now send 200 requests in total with a less suspicious FPIB value of 0.2; this reduced suspiciousness could help the malicious $\mathcal{L}$ blend into the benign $\mathcal{L}$ sets and avoid detection.

(3) Blending in by redistributing the guess set. The attacker could also attempt to blend into background benign traffic by redistributing their guess set across additional IP addresses (by changing the IP address fields in their attack transcript). For example, consider the case of detecting targeted behavior using the AUP feature. If an attack $\mathcal{L}$ exhibits targeted attack behavior by sending 25 different password guesses to a single user, then it would have a AUP of 25 and would likely be detected for such a high AUP value. However, the attacker can evade detection by sending these 25 different password guesses from 25 different $\mathcal{L}$ sets. Each of these 25 $\mathcal{L}$ sets would now have an AUP of one, effectively avoiding detection, as an AUP value of one does not exhibit targeted behavior. For the sake

of evaluation, we assume that the attacker evenly splits the total number of requests and the number of requests exhibiting any suspicious feature across all $\mathcal{L}$ sets.

(4) Blending in by slowing the attack. Finally, an attacker can insert additional delay between successive login requests to minimize the suspiciousness of the timestamp-based features (i.e., MIT, SIT). This strategy is effective for evading malicious $\mathcal{L}$ sets that send a large number of requests within a short period of time. For example, consider an attacker sending NR login requests from an IP within t seconds. Assuming the attacker is evenly distributing these NR logins over the time period t, the attack $\mathcal{L}$ would have a mean interarrival time of MIT $=$ t/NR. By remaining active for an additional θ_t period, the attacker could increase (and thus reduce suspiciousness of) its MIT feature value to $(t + \theta_t)$/NR. Attack tools (e.g., OpenBullet2 [153]) typically provide such an option to reschedule logins to a given interval. Of course, this would increase the cost for the attacker, requiring them to hold the IP addresses for a longer period of time.

Evasion results. To investigate the efficacy of the evasion strategies, we explore a set of reasonable amounts of resources an evasion attacker could use for evading the attack clusters: $\theta_{ip} \in \{10, 20, 35\}$, $\theta_r = 1,000$, and $\theta_t = 2$ hours. The per-day Araña is relatively robust to evasion as it uses clusters of requests, which is harder to evade. We found that by using specially crafted chaff $\mathcal{L}$ sets, an attacker can evade some types of attacks at high cost. For example, with an extra $\theta_{ip} = 35$ IP addresses, $\theta_r = 1000$ additional requests, and $\theta_t = 2$ hours, it is possible to evade 3 out of 29 attack clusters.

C SUPPLEMENTARY MATERIAL: CHAPTER 5

C.1 Two Common Strategies of PMS for Credential Backup Protection

As discussed in Section 5.2, PMS providers typically offer users two strategies to secure their credential backups with user secrets. One is based on *key derivation* and the other is based on *key escrow*.

Key derivation. In the first strategy, a user's credential backup is encrypted using a key that only the user can access. The encryption key can be directly derived from a user-chosen secret (e.g., a "master password"), e.g., via key derivation functions (KDF). This approach provides end-to-end encryption to secure users' passkeys — encryption and decryption of a user's PMS passkey backup are performed locally on the user's authenticator, avoiding giving PMS access to the user-chosen secret and the derived encryption key. This strategy is adopted by popular credential managers such as LastPass [9], 1Password [8]. While this strategy gives the user some security control over their passkeys backup at PMS provider, it also enables attackers to perform offline cracking on the user's passkey backup when it gets leaked from the provider. In particular, prior research has observed that user-chosen secrets are easily guessable, especially those that users can recall consistently [12]. Many applications utilize the KDF defined in PKCS#12 which is not appropriate to withstand offline attacks leveraging modern hardware [154]. Thus, ensuring the protection of users' passkeys with this strategy has long been a significant challenge. For example, serious concerns have been raised regarding the impact of offline cracking attacks following the breach of LastPass's cloud storage [155, 156].

Key escrow. The second strategy requires the user secret (e.g., a PIN or

passcode, or a screen-lock pattern) to be independent of the encryption key generation and serves solely as a "verification secret" that the PMS provider or its components use to verify the user's identity. Specifically, provider independently generates an encryption key to encrypt a user's passkey backup at rest and stores the key on its own key management service (KMS). When the user wishes to retrieve their credential backup, the PMS provider retrieves their key from KMS securely only if the user presents a valid "verification secret". This strategy is adopted by iCloud KeyChain from Apple, Google Password Manager, and Password Monitor from Microsoft [157, 158, 159]. Compared to the former strategy, this strategy requires a user to give up their control over their credential backups and trust that the provider can implement their cloud storage and KMS correctly and securely as expected, even though this may not always be the case [160].

Furthermore, even if PMS implementation is secure, insider attackers within the PMS provider could potentially obtain query access to the KMS and retrieve the key after successfully guessing the low-entropy user verification secret [108, 109]. This remains possible even under the provider's rate-limiting policies [161, 159], which could not only fail to prevent such attacks but also undesirably introduce denial-of-service concerns for users. What is more concerning is that many PMS providers adopt one or more non-cryptographic authentication methods such as passwords or secret questions for account access, recovery, or verification secret reset [162]. In this case, the security of the user's passkey backups against a remote attacker may eventually fall back to that of those weaker authentication methods.

C.2 Definition of KDF

A key derivation function (KDF) is a fundamental cryptographic primitive that produces cryptographic keys from a private input, such as a user password. When used as encryption keys for data storage or transmission, it is crucial that the keys generated by a KDF are computationally indistinguishable from random strings [163] in order to prevent an attacker from obtaining useful information of the private input string. In other words, an attacker should not be able to determine whether a given binary string is a cryptographic key produced by a KDF or just a random string of equivalent length. In this chapter, we follow standard assumptions by requiring a KDF to be a pseudorandom function (PRF) [163] and consider the following definition of a KDF:

Definition C.1 (Key Derivation Function)**.** *A key derivation function (KDF), denoted by* $\mathsf{KDF}(w, z)$*, is a pseudorandom function F:* $\{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$ *that takes as input a user detection secret* w *and a randomness* z *uniformly chosen from* $\{0,1\}^\kappa$ *and outputs a user key* u*.*

C.3 Flatness Preservation

Here we show that the flatness of detection secrets used in CASPER is *preserved* despite the compromise of PMS storage. To capture how accurately a distinguishing attacker can identify w_{i^*} from W output by $\mathcal{G} = \langle \mathsf{GenDetectSecrets}, \mathsf{SelectRealSecret} \rangle$, we consider a *flatness* experiment $\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}$ defined as follows:

$$
\begin{array}{l}
\texttt{Experiment } \mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D}) \\
\quad W \xleftarrow{\$} \mathsf{GenDetectSecrets}(k) \\
\quad w_{i^*} \leftarrow \mathsf{SelectRealSecret}(W, \eta) \\
\quad \hat{w} \leftarrow \mathcal{D}_{\mathcal{G}}(W) \\
\quad \texttt{if } \hat{w} = w_{i^*} \\
\quad\quad \texttt{then return } 1 \\
\quad \texttt{else return } 0
\end{array}
$$

We define the advantage of $\mathcal{D}$, given η and k, as:

$$
\begin{aligned}
\mathsf{Adv}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D}) &\stackrel{\text{def}}{=} \mathbb{P}\left(\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D}) = 1\right) - \frac{1}{k+1}, \\
\mathsf{Adv}^{\mathsf{flt},\mathcal{G}}_{\eta,k} &\stackrel{\text{def}}{=} \max_{\mathcal{D}}\{\mathbb{P}\left(\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D}) = 1\right)\},
\end{aligned}
$$

where the maximum is taken over all distinguishing attackers $\mathcal{D}$.

We then consider the attacker's ability to get a user's passkey backup entries for *multiple* accounts leaked from PMS by defining the following PMS oracle $\mathcal{O}_{\mathsf{cbs}}$ that takes w_{i^*} as input and outputs a pair $(\tilde{s}, z)$ for each oracle query. Then we consider the following experiment to characterize how much better a decoy distinguishing adversary $\mathcal{A}$ can distinguish w_{i^*} when it additionally has access to such a PMS oracle $\mathcal{O}_{\mathsf{cbs}}$, where aid, uid and sid are ignored due to the independence of W and w_{i^*} on them:

$$
\begin{array}{l}
\text{Oracle } \mathcal{O}_{\mathsf{cbs}}(w_{i^*}) \\
\quad s_{i^*} \xleftarrow{\$} \{0,1\}^{\kappa} \\
\quad (\tilde{s}, z) \leftarrow \Pi_{\mathsf{EncCred}}(w_{i^*}, s_{i^*}) \\
\quad \texttt{return } (\tilde{s}, z)
\end{array}
$$

$$
\begin{array}{l}
\texttt{Experiment } \mathsf{Expt}^{\mathsf{flt,cbs}}_{\eta,k}(\mathcal{A}) \\
\quad W \xleftarrow{\$} \mathsf{GenDetectSecrets}(k) \\
\quad w_{i^*} \leftarrow \mathsf{SelectRealSecret}(W, \eta) \\
\quad \hat{w} \leftarrow \mathcal{A}_{\mathcal{G},\mathcal{O}_{\mathsf{cbs}}(w_{i^*})}(W) \\
\quad \texttt{if } \hat{w} = w_{i^*} \\
\quad\quad \texttt{then return } 1 \\
\quad \texttt{else return } 0
\end{array}
$$

We define the advantage of $\mathcal{A}$ as

$$
\mathsf{Adv}^{\mathsf{flt,cbs}}_{\eta,k}(\mathcal{A}) \stackrel{\text{def}}{=} \mathbb{P}\left(\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{A}) = 1\right) - \frac{1}{k+1}
$$

$$
\mathsf{Adv}^{\mathsf{flt,cbs}}_{\eta,k} \stackrel{\text{def}}{=} \max_{\mathcal{A}}\{\mathsf{Adv}^{\mathsf{flt,cbs}}_{\eta,k}(\mathcal{A})\},
$$

where the maximum is taken over all flatness adversaries $\mathcal{A}$.

Proposition C.3.1.

$$
\mathsf{Adv}^{\mathsf{flt},\mathit{cbs}}_{\eta,k} = \mathsf{Adv}^{\mathsf{flt},\mathcal{G}}_{\eta,k}.
$$

Proof: Given $\mathcal{A}$ for the experiment $\mathsf{Expt}^{\mathsf{flt,cbs}}_{\eta,k}$, we construct a decoy distinguisher $\mathcal{D}$ for the experiment $\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}$ defined in Section 5.4. $\mathcal{D}$ provides $\mathcal{A}$ with W it receives from the experiment. $\mathcal{D}$ responds to $\mathcal{A}$'s $\mathcal{G}$ query by its own $\mathcal{G}$ query response. For each $\mathcal{O}_{\mathsf{cbs}}$ oracle query made by $\mathcal{A}$, $\mathcal{D}$ does the following:

- $\mathcal{D}$ chooses $\tilde{s}'$ and z' from $\{0,1\}^\kappa$ uniformly at random.
- For all $w_i \in W$, $\mathcal{D}$ runs $s'_i \leftarrow \Pi_{\mathsf{DecCred}}(w_i, \tilde{s}', z')$ and check if s'_i is a valid passkey, i.e., $s'_i \stackrel{?}{\in} \mathcal{S}$, where $\mathcal{S}$ is the passkey space.
  - If there exists $w_i \in W$ such that $s'_i \notin \mathcal{S}$, $\mathcal{D}$ restarts the whole process by re-choosing a fresh $\tilde{s}'$ uniformly at random.

– If $s'_i \in \mathcal{S}$ for all $w_i \in W$, $\mathcal{D}$ returns $(\tilde{s}', z')$ to $\mathcal{A}$

Note that this process is efficient because it is almost unlikely that an invalid passkey will be produced (see Appendix C.4). Finally $\mathcal{D}$ outputs 1 if $\mathcal{A}$ outputs 1. Considering that, given a uniformly randomly chosen z', for each $w_i \in W$, there exists a corresponding s_i uniformly distributed in $\mathcal{S}$ such that $(\tilde{s}', z') = \mathsf{KDF}(w_i, z') \oplus s_i$, and so $(\tilde{s}', z')$ and $(\tilde{s}, z)$ are distributed identically for each $\mathcal{O}_{\mathsf{cbs}}$ oracle query simulated by $\mathcal{D}$, we have:

$$\mathbb{P}\left(\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D}) = 1\right) \geqslant \mathbb{P}\left(\mathsf{Expt}^{\mathsf{flt},\mathsf{cbs}}_{\eta,k}(\mathcal{A}) = 1\right),$$
$$\mathsf{Adv}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D}) \geqslant \mathsf{Adv}^{\mathsf{flt},\mathsf{cbs}}_{\eta,k}(\mathcal{A}).$$

Also, we can construct a $\mathcal{A}$ for the experiment who runs a decoy distinguisher $\mathcal{D}$ for the experiment $\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}$ as a subroutine. Upon receiving W from the experiment, $\mathcal{A}$ directly provides W as $\mathcal{D}$'s input and responds to $\mathcal{D}$'s $\mathcal{G}$ query by its own $\mathcal{G}$ query response. Finally $\mathcal{A}$ outputs 1 if $\mathcal{D}$ outputs 1 so:

$$\mathbb{P}\left(\mathsf{Expt}^{\mathsf{flt},\mathsf{cbs}}_{\eta,k}(\mathcal{A}) = 1\right) \geqslant \mathbb{P}\left(\mathsf{Expt}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D}) = 1\right),$$
$$\mathsf{Adv}^{\mathsf{flt},\mathsf{cbs}}_{\eta,k}(\mathcal{A}) \geqslant \mathsf{Adv}^{\mathsf{flt},\mathcal{G}}_{\eta,k}(\mathcal{D}).$$

C.4 Well-Formed ECDSA Keys from Decrypting $\tilde{s}$ with an Incorrect w

Here we show that decrypting $\tilde{s}$ with an incorrect detection secret w, where $w \neq w_{i^*}$, almost certainly results in a well-formed private key for ECDSA.

For elliptic curves supported by FIDO2 for ECDSA, e.g., secp256r1 and secp256k1, n is the order of the base point of the curve and is a large prime number. Specifically, n is equal to `0xFFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551` and `0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEBAAEDCE6AF48A03BBFD25E8CD0364141` for secp256r1 and secp256k1 respectively [164]. Decrypting $\tilde{s}$ with an incorrect w will yield s ($\neq s_{i^*}$) that is uniformly distributed in the range of $[0, 2^{256}-1]$ (except for s_{i^*}), given w is uniformly sampled by GenDetectSecrets (see Figure 5.2). For s ($\neq s_{i^*}$) to be a valid private key of ECDSA on secp256r1 or secp256k1, s needs to be within the range of $[1, n-1]$. Thus, the probability that decrypting $\tilde{s}$ with an incorrect detection secret w results in a well-formed private key is $\frac{n-2}{2^{256}-1} > 0.999999999767$ for ECDSA on secp256r1 and $\frac{n-2}{2^{256}-1} > 0.999999999999$ on secp256k1. On another note, if decrypting $\tilde{s}$ with an incorrect key is non-negligibly unlikely to produce a well-formed private key for a given signature scheme, an alternative design could be adopted, as mentioned in Section 5.5. Specifically, instead of producing and syncing a single $\tilde{s}$, one could randomly generate k additional key pairs as decoys and synchronize $k+1$ private keys (passkeys) to the PMS directly.

C.5 More about User Secret η

Here, we first explore alternative methods for instantiating the user secret η without imposing a memory burden on users. We then discuss potential privacy concerns related to η.

User memory-independent η. In practice, there are other desirable ways to instantiate η, making it user memory-independent but retrievable by users from physical objects (e.g., using credit cards CVVs as η retrievable from credit cards) or from third parties trusted for maintaining the secrecy and availability of the secrets (e.g., using several digits of bank account numbers as η retrievable from banks). Additionally, η can be derived

from users' biometric data (e.g., fingerprints or faces) [165] . While such instantiation of η adds little memory burden on users and raises negligible privacy concerns as discussed next when w_{i^*} is stolen, the user interface should clearly explain how CASPER uses these secrets and, more importantly, that η will never be stored at any participating party in CASPER.

Privacy of η A `caat` who has compromised W and luckily guessed the correct w_{i^*} might be tempted to deduce η from W and w_{i^*}. However, this is in fact not easy because the modulo operation in SelectRealSecret restricts the information leakage about η to only $\log_2(k+1)$ bits of entropy. For example, considering the case where a user's η is the last four digits of their bank account number and k is set to 32, η would be hidden from the attacker among $10000 \times \frac{1}{32+1} \approx 303$ different 4-digit numbers that, if taken with W by SelectRealSecret as input, would yield the same w_{i^*}.

An alternative design for generating W given η. Recall that in Figure 5.2, we provided an example of generating W and determining the real secret index based on a user input η. Here, we introduce an alternative design for generating W that reduces the probability of a manual η input error leading to a false detection. The concept is straightforward: CASPER can assemble a set (denoted as N) of size k, which includes decoy user secrets randomly sampled from the same secret space of η. For example, when $k = 32$ and η is an x-digit system-generated PIN where $2 \leqslant x < \log_{10} 2^{\kappa}$, CASPER can randomly select 32 distinct PINs as the decoy user secrets from the remaining $10^x - 1$ (incorrect) PINs. CASPER then directly assembles W by including $\mathsf{Hash}(\eta)$ as the real detection secret and the hashes of the k decoy user secrets in N as decoys. This design provides better resistance against false detection triggered by manual η user input errors. In the x-digit PIN example here, a randomly entered incorrect η' will be mapped to a decoy passkey with a probability of at most $\frac{32}{10^x}$. This design is more suitable when η input errors are a larger concern than the privacy of η, as the `caat` can more easily determine η if it identifies the real detection

- Upon receiving a login request for uid, RP retrieves $\mathsf{v}(=\mathsf{s})$ for uid and requests an OTP, if the uid exists.
- User device runs $w \leftarrow \mathsf{GenOTP}(\mathsf{s})$ and sends wto RP.
- Upon receiving wfrom the user, RP performs the following tests and actions:
  - `if` $\forall \mathsf{v} \in V : w \neq \mathsf{GenOTP}(\mathsf{v})$: RP rejects this login request.
  - `else if` $\mathsf{v} \in V'$, RP raises a detection alarm.
  - `else` RP accepts this login request.

Figure C.1: The compromise detection algorithm of CASPER for OTP. The BnR protocol remains the same for OTP.

secret $(= \mathsf{Hash}(\eta))$ within W.

C.6 Extending CASPER for One Time Passwords

Besides passkeys, the concept of CASPER can also be easily extended to authentication methods based on other cryptographic credentials such as HMAC or time-based one-time passwords (OTP [166, 167]).

Background on OTP. Unlike FIDO2, OTP authentication is based on a shared secret between a user device (or authenticator) and an RP (the authentication server). During registration, the user device and the RP agree on a shared cryptographic seed s sampled uniformly at random. During the login phase, the user device generates a new OTP from s as $w \leftarrow \mathsf{GenOTP}(\mathsf{s})$. Here GenOTP takes s as its input and outputs an OTP that is a truncated output of $\mathsf{HMAC}(\mathsf{s}, \mathsf{Count})$ (as defined in [166, Sec. 5.2] and [167, Sec. 1.2]) where Count is a counter used to maintain the freshness

of a new OTP. Thus, Countis updated either incrementally for each login (i.e., HMAC-based OTP [166]) or periodically based on the current time (i.e., timing-based OTP [167]). When the user submits w to the RP for authentication, the RP also generates an OTP w′ by invoking GenOTP with the shared s and the same counter Count, and verifies the user's identity by checking $w \stackrel{?}{=} w'$.

Extending CASPER for OTP. The BnR protocol for OTP is similar to the one for FIDO2 as shown in Figure 5.3. As shown in Figure C.1, the compromise detection algorithm here is largely similar to the one used for FIDO2-based authentication, with only slight variations due to differences in their authentication flows. During a login, the user device generates an OTP via $w \leftarrow \mathsf{GenOTP}(\mathsf{s}_{i^*})$, and sends the OTP w to the RP. Note that for OTP, there is no privacy and public key pair, and in a credential pair $\langle s, v \rangle$, $s = v$ because they are the same random seed shared and kept private by both the user device and RP. However, we keep the notations of s and v just to follow the specification of our generic BnR protocol. Therefore, to authenticate the login request, the RP will also generate an OTP w' on their own for each v in V from the list of verifiers , and check if $w' = w$. Similar to the detection algorithm for FIDO2 as described above, if there exists $\mathsf{v} \in V$ such that $w' = w$, then whether v is in the active verifier set V' or not determines if this login attempt will trigger a PMS detection or result in a successful login.

Handling spoofed false alert notifications from RP for OTP. Unlike FIDO2, detecting spoofed false alerts with a verifiable proof is challenging for OTP-based authentication methods, or, more broadly, authentication methods based on shared secrets between the user and the RP. This is because both parties can generate the same authentication response from their shared secret, making it difficult to determine whether a triggering authentication response is produced by a PMS breaching attacker or a misbehaving RP itself.

While future work can investigate further how to handle this challenge, for now, we propose an alternative approach that instead provides a *probabilistic* guarantee against such alerts. This approach requires the user device to select a subset of W at random, denoted by W', while still using W as the input of $\Pi_{\mathsf{GenVerifierSet}}$ to generate V for the RP. Meanwhile, CASPER requires the RP to send back the triggering OTP was a probabilistic detection proof. With this setup, the probability of a RP producing a wwith an OTP seed, s_i, that corresponds to a detection secret $w \in W \setminus W'$ is $1 - \frac{|W'|}{|W|}$. When this happens, the user can learn that the RP intentionally raised a false detection.

BIBLIOGRAPHY

[1] Verizon Enterprise. 2020 Databreach Investigation Report. https://enterprise.verizon.com/resources/reports/2020/2020-data-breach-investigations-report.pdf, 2020.

[2] CNBC: Why passkeys from Apple, Google, Microsoft may soon replace your passwords. https://fidoalliance.org/cnbc-why-passkeys-from-apple-google-microsoft-may-soon-replace-your-passwords/, 2024.

[3] Junade Ali. Optimising caching on pwned passwords (with workers). https://blog.cloudflare.com/optimising-caching-on-pwnedpasswords, 2018.

[4] One Stolen Password Took Down The Colonial Pipeline — Is Your Business Next? https://www.forbes.com/sites/forbestechcouncil/2021/09/14/one-stolen-password-took-down-the-colonial-pipeline---is-your-business-next.

[5] Apple. About the security of passkeys. 2024. https://support.apple.com/en-us/HT213305l.

[6] Passwordless login with passkeys. https://developers.google.com/identity/passkeys, June 2023.

[7] Passkeys in Windows. https://support.microsoft.com/en-us/windows/passkeys-in-windows-301c8944-5ea2-452b-9886-97e4d2ef4422, Available on: 2025-01-30.

[8] Unlock 1Password with a passkey (beta). https://support.1password.com/passkeys/, 2024.

[9] Start Your Passwordless Authentication Journey - LastPass. https://www.lastpass.com/features/passwordless-authentication, 2024.

[10] Passkeys in Dashlane. https://support.dashlane.com/hc/en-us/articles/7888558064274-Passkeys-in-Dashlane, 2024.

[11] Marina Sanusi Bohuk, Mazharul Islam, Syed Suleman Ahmad, Michael Swift, Thomas Ristenpart, and Rahul Chatterjee. Gossamer: Securely Measuring Password-based Logins. In *USENIX Security 22*. USENIX Association, 2022.

[12] M. Islam, M. S. Bohuk, P. Chung, T. Ristenpart, and R. Chatterjee. Araña: Discovering and characterizing password guessing attacks in practice. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1019–1036, Anaheim, CA, August 2023. USENIX Association.

[13] OAuth. https://oauth.net/2/, 2020.

[14] FIDO Alliance. Client to authenticator protocol (ctap). https://fidoalliance.org/specs/fido-v2.2-rd-20230321/fido-client-to-authenticator-protocol-v2.2-rd-20230321.html, 2023.

[15] W3C. Web authentication: An api for accessing public key credentials level 3. https://www.w3.org/TR/webauthn-3/, 2023.

[16] M. Barbosa, A. Boldyreva, S. Chen, and B. Warinschi. Provable security analysis of FIDO2. In *Advances in Cryptology–CRYPTO 2021: 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16–20, 2021, Proceedings, Part III 41*, pages 125–156. Springer, 2021.

[17] N. Bindel, C. Cremers, and M. Zhao. FIDO2, CTAP 2.1, and WebAuthn 2: Provable security and post-quantum instantiation. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 1471–1490. IEEE, 2023.

[18] J. Guan, H. Li, H. Ye, and Z. Zhao. A formal analysis of the FIDO2 protocols. In *European Symposium on Research in Computer Security*, pages 3–21. Springer, 2022.

[19] Two-Factor Authentication Market. https://www.marketresearchfuture.com/reports/two-factor-authentication-market-3772, Available on: 2024-03-01.

[20] K. Owens, O. Anise, A. Krauss, and B. Ur. User Perceptions of the Usability and Security of Smartphones as FIDO2 Roaming Authenticators. In *SOUPS USENIX Security Symposium*, pages 57–76, 2021.

[21] L. Lassak, E. Pan, B. Ur, and M. Golla. Why Aren't We Using Passkeys? Obstacles Companies Face Deploying FIDO2 Passwordless Authentication. In *Proceedings of the 33rd USENIX Security Symposium. Philadelphia, PA, August 2024*, 2024.

[22] M. Kepkowski, M. Machulak, I. Wood, and D. Kaafar. Challenges with passwordless fido2 in an enterprise setting: A usability study. In *2023 IEEE Secure Development Conference (SecDev)*, pages 37–48. IEEE, 2023.

[23] L. Würsching, F. Putz, S. Haesler, and M. Hollick. FIDO2 the Rescue? Platform vs. Roaming Authentication on Smartphones. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2023.

[24] S. G. Lyastani, M. Schilling, M. Neumayr, M. Backes, and S. Bugiel. Is FIDO2 the kingslayer of user authentication? A comparative usability study of FIDO2 passwordless authentication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 268–285. IEEE, 2020.

[25] F. M. Farke, L. Lorenz, T. Schnitzler, P. Markert, and M. Dürmuth. "You still use the password after all"—exploring FIDO2 security keys in a small company. In *Sixteenth Symposium on Usable Privacy and Security (SOUPS 2020)*, pages 19–35, 2020.

[26] L. Lassak, A. Hildebrandt, M. Golla, and B. Ur. " it's stored, hopefully, on an encrypted server": Mitigating users' misconceptions about FIDO2 biometric WebAuthn. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 91–108, 2021.

[27] K. Bicakci and Y. Uzunay. Is FIDO2 Passwordless Authentication a Hype or for Real?: A Position Paper. In *2022 15th International Conference on Information Security and Cryptography (ISCTURKEY)*, pages 68–73. IEEE, 2022.

[28] E. Ulqinaku, H. Assal, A. AbdelRahman, S. Chiasson, and S. Capkun. Is Real-time Phishing Eliminated with FIDO? Social Engineering Downgrade Attacks against FIDO Protocols. In *Proceedings of the 30th USENIX Security Symposium (USENIX Security 21)*, pages 3811–3828. USENIX Association, 2021.

[29] E. Naprys. Third of Americans use password managers. `https://cybernews.com/security/third-of-americans-use-password-managers/`, 2024.

[30] J. Doolani, M. Wright, R. Setty, and S. M. Haque. Locimotion: Towards learning a strong authentication secret in a single session. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–13, 2021.

[31] Grant Ho, Aashish Sharma, Mobin Javed, Vern Paxson, and David Wagner. Detecting Credential Spearphishing Attacks in Enterprise Settings. In *USENIX Security 17*. USENIX Association, 2017.

[32] J. Bonneau. The science of guessing: Analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE Symposium on Security and Privacy*, pages 538–552, 2012.

[33] Rahul Chatterjee, Anish Athalye, Devdatta Akhawe, Ari Juels, and Thomas Ristenpart. pASSWORD tYPOS and how to correct them securely. 2016. Full version of the paper can be found at https://cs.cornell.edu/ rahul/papers/pwtypos.pdf.

[34] Andrew Miklas, Stefan Saroiu, Alec Wolman, and Angela Demke Brown. Bunker: A privacy-oriented platform for network tracing. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2009.

[35] Dan Lowe Wheeler. zxcvbn: Low-Budget Password Strength Estimation. In *Proc. USENIX Security*, 2016.

[36] Bijeeta Pal, Mazharul Islam, Marina Sanusi Bohuk, Nick Sullivan, Luke Valenta, Tara Whalen, Christopher Wood, Thomas Ristenpart, and Rahul Chatterjee. Might I Get Pwned: A Second Generation Compromised Credential Checking Service. In *USENIX Security 22*. USENIX Association, 2022.

[37] William Melicher, Blase Ur, Sean M. Segreti, Saranga Komanduri, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Fast, lean, and accurate: Modeling password guessability using neural networks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 175–191, Austin, TX, 2016. USENIX Association.

[38] Matt Weir, Sudhir Aggarwal, Breno de Medeiros, and Bill Glodek. Password Cracking Using Probabilistic Context-Free Grammars. In *2009 30th IEEE Symposium on Security and Privacy*, 2009.

[39] Arvind Narayanan and Vitaly Shmatikov. Fast Dictionary Attacks on Passwords Using Time-Space Tradeoff. In *12th ACM Conference on Computer and Communications Security*, CCS '05, page 364–372, New York, NY, USA, 2005. Association for Computing Machinery.

[40] Briland Hitaj, Paolo Gasti, Giuseppe Ateniese, and Fernando Perez-Cruz. PassGAN: A Deep Learning Approach for Password Guessing. In *International conference on applied cryptography and network security*, pages 217–237. Springer, 2019.

[41] Jerry Ma, Weining Yang, Min Luo, and Ninghui Li. A study of probabilistic password models. In *2014 IEEE Symposium on Security and Privacy*, 2014.

[42] Saranga Komanduri, Richard Shay, Patrick Gage Kelley, Michelle L. Mazurek, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, and Serge Egelman. Of passwords and people: Measuring the effect of password-composition policies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, page 2595–2604, New York, NY, USA, 2011. Association for Computing Machinery.

[43] Blase Ur, Patrick Gage Kelley, Saranga Komanduri, Joel Lee, Michael Maass, Michelle L. Mazurek, Timothy Passaro, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. How does your password measure up? the effect of strength meters on password creation. In *Proceedings of the 21st USENIX Conference on Security Symposium*, Security'12, page 5, USA, 2012. USENIX Association.

[44] P. G. Kelley, S. Komanduri, M. L. Mazurek, R. Shay, T. Vidas, L. Bauer, N. Christin, L. F. Cranor, and J. Lopez. Guess again (and again and again): Measuring password strength by simulating password-cracking algorithms. In *2012 IEEE Symposium on Security and Privacy*, pages 523–537, 2012.

[45] Hana Habib, Jessica Colnago, William Melicher, Blase Ur, Sean Segreti, Lujo Bauer, Nicolas Christin, and Lorrie Cranor. Password creation in the presence of blacklists. *Proc. USEC*, page 50, 2017.

[46] Dinei Florencio and Cormac Herley. A large-scale study of web password habits. In *Proceedings of the 16th International Conference on World Wide Web*, WWW '07, pages 657–666, New York, NY, USA, 2007. ACM.

[47] Alain Forget, Saranga Komanduri, Alessandro Acquisti, Nicolas Christin, Lorrie Faith Cranor, and Rahul Telang. Building the security behavior observatory: An infrastructure for long-term monitoring of client machines. In *Proceedings of the 2014 Symposium and Bootcamp on the Science of Security*, HotSoS '14, New York, NY, USA, 2014. Association for Computing Machinery.

[48] Sarah Pearman, Jeremy Thomas, Pardis Emami Naeini, Hana Habib, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Serge Egelman, and Alain Forget. Let's go in for a closer look: Observing passwords in their natural habitat. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 295–310, New York, NY, USA, 2017. Association for Computing Machinery.

[49] Michelle L. Mazurek, Saranga Komanduri, Timothy Vidas, Lujo Bauer, Nicolas Christin, Lorrie Faith Cranor, Patrick Gage Kelley, Richard Shay, and Blase Ur. Measuring Password Guessability for an Entire University. In *2013 ACM SIGSAC Conference on Computer and Communications Security* (*CCS '13*). Association for Computing Machinery, 2013.

[50] D. Harkings. Synthetic initialization vector (SIV) authenticated encryption using the advanced encryption standard (AES). `https://tools.ietf.org/html/rfc5297#section-2.6`.

[51] Latanya Sweeney. Weaving technology and policy together to maintain confidentiality. *The Journal of Law, Medicine & Ethics*, 25(2-3):98–110, 1997.

[52] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 111–125. IEEE, 2008.

[53] Python fernet cryptography library. `https://cryptography.io/en/latest/fernet.html`.

[54] Python miscreant encryption library. `https://github.com/miscreant/miscreant.py`.

[55] RockYou hack: From bad to worse. `https://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords/`, 2009.

[56] Best64 rule list. `https://github.com/hashcat/hashcat/blob/master/rules/best64.rule`, 2018.

[57] Jens Steube and Gabriele Gristina. Hashcat. `https://hashcat.net/hashcat/`.

[58] WAQAS. A trove of 1.4 billion clear text credentials file found on dark web. `https://www.hackread.com/billion-of-credentials-found-on-dark-web/`, 2017.

[59] Tara Seals. Billions of passwords offered for $2 in cyber-underground. *ThreatPost*, 2021.

[60] Bijeeta Pal, Tal Daniel, Rahul Chatterjee, and Thomas Ristenpart. Beyond credential stuffing: Password similarity models using neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.

[61] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. Targeted online password guessing: An underestimated threat. In *2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. Association for Computing Machinery, 2016.

[62] Anupam Das, Joseph Bonneau, Matthew Caesar, Nikita Borisov, and XiaoFeng Wang. The tangled web of password reuse. In *NDSS*, volume 14, pages 23–26, 2014.

[63] Lucy Li, Bijeeta Pal, Junade Ali, Nick Sullivan, Rahul Chatterjee, and Thomas Ristenpart. Protocols for checking compromised credentials. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, page 1387–1403, New York, NY, USA, 2019. Association for Computing Machinery.

[64] Mobin Javed and Vern Paxson. Detecting stealthy, distributed ssh brute-forcing. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 85–96, 2013.

[65] Erin Kenneally and David Dittrich. The menlo report: Ethical principles guiding information and communication technology research. *Available at SSRN 2445102*, 2012.

[66] Chris Evans, Chris Palmer, and Ryan Sleevi. Public key pinning extension for HTTP. *Internet Engineering Task Force. 27Available: http://www. ietf. org/internet-drafts/draft-ietf-websec-key-pinning-09. txt*, 2015.

[67] MaxMind. GeoIP2 Database. `https://www.maxmind.com/en/geoip2-databases`.

[68] LLC MaxMind. GeoIP. https://www.maxmind.com/en/geoip-demo.

[69] Inwi – opérateur téléphonique mobile, fixe & internet au maroc. `https://www.inwi.ma/`.

[70] Google Cloud: Cloud Computing Services. `https://cloud.google.com/`.

[71] Danna Thee. Sentry MBA. `https://e.cyberint.com/hubfs/Sentry%20MBA%20Report/CyberInt_Sentry-MBA_Report.pdf`.

[72] DigitalOcean – the developer cloud. `https://www.digitalocean.com/`.

[73] John Franks, Phillip Hallam-Baker, Jeffrey Hostetler, Scott Lawrence, Paul Leach, Ari Luotonen, and Lawrence Stewart. HTTP authentication: Basic and digest access authentication, 1999.

[74] D Freeman, Sakshi Jain, Markus Duermuth, Battista Biggio, and Giorgio Giacinto. Who are you? a statistical approach to measuring user authenticity. In *NDSS*, 2016.

[75] Richard Shay, Patrick Gage Kelley, Saranga Komanduri, Michelle L. Mazurek, Blase Ur, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Correct horse battery staple: Exploring the usability of system-assigned passphrases. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, SOUPS '12, New York, NY, USA, 2012. Association for Computing Machinery.

[76] Richard Shay, Saranga Komanduri, Adam L. Durity, Phillip (Seyoung) Huh, Michelle L. Mazurek, Sean M. Segreti, Blase Ur, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. Can long passwords be secure and usable? In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, page 2927–2936, New York, NY, USA, 2014. Association for Computing Machinery.

[77] Duo Security: Two-Factor Authentication & End Point Security. `https://duo.com/`, 2021.

[78] Suzanne Widup, Bob Rudis, Dave Hylender, and Marc Spitler. 2015 Data Breach Investigations Report. `https://www.researchgate.net/publication/289254638_2015_Verizon_Data_Breach_Investigations_Report`, 2015.

[79] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, Angelika Moscicki, Daniel Margolis, Vern Paxson, and Elie Bursztein, editors. *Data breaches, phishing, or malware? Understanding the risks of stolen credentials*, 2017.

[80] Cormac Herley and Stuart E. Schechter. Distinguishing Attacks from Legitimate Authentication Traffic at Scale. In *Network and Distributed Systems Security (NDSS) Symposium 2019*, 2019.

[81] Daniel Müllner. Modern hierarchical, agglomerative clustering algorithms. *arXiv preprint arXiv:1109.2378*, 2011.

[82] Troy Hunt. Have I Been Pwned? `https://haveibeenpwned.com/Passwords/`, 2018.

[83] Michael Tremante. End User Security: Account Takeover Protections with Cloudflare. `https://blog.cloudflare.com/account-takeover-protection/`, 2021.

[84] `https://github.com/islamazhar/Arana-Public`.

[85] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. The quest to replace passwords: A framework for comparative evaluation of web authentication schemes. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 553–567. IEEE, 2012.

[86] Rahul Chatterjee, Anish Athayle, Devdatta Akhawe, Ari Juels, and Thomas Ristenpart. pASSWORD tYPOS and how to correct them securely. In *2016 IEEE Symposium on Security and Privacy*, 2016.

[87] Openwall. John The Ripper. `https://www.openwall.com/john/`.

[88] Saranga Komanduri. Modeling the Adversary to Evaluate Password Strength With Limited Samples. `https://kilthub.cmu.edu/articles/thesis/Modeling_the_Adversary_to_Evaluate_Password_Strength_With_Limited_Samples/6720701/1`, 2018.

[89] Kurt Thomas, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, Tadek Pietraszek, Sarvar Patel, Dan Boneh, and Elie Bursztein. Protecting accounts from credential stuffing with password breach alerting. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1556–1571, Santa Clara, CA, 2019. USENIX Association.

[90] Joseph Bonneau, Cormac Herley, Paul C Van Oorschot, and Frank Stajano. Passwords and the evolution of imperfect authentication. *Communications of the ACM*, 58(7), 2015.

[91] S. Schechter, Y. Tian, and C. Herley. Stopguessing: Using guessed passwords to thwart online guessing. In *2019 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 576–589, 2019.

[92] Jessica Colnago, Summer Devlin, Maggie Oates, Chelse Swoopes, Lujo Bauer, Lorrie Cranor, and Nicolas Christin. "It's not actually that horrible": Exploring Adoption of Two-Factor Authentication at a University. In *2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. Association for Computing Machinery, 2018.

[93] Katie Deighton. Tech Companies Push Users to Adopt Two-Factor Authentication. `https://www.wsj.com/articles/tech-companies-push-users-to-adopt-two-factor-authentication-11635807088`, 2021.

[94] Periwinkle Doerfler, Kurt Thomas, Maija Marincenko, Juri Ranieri, Yu Jiang, Angelika Moscicki, and Damon Mccoy. Evaluating Login Challenges as a Defense Against Account Takeover. In *28th International Conference on World Wide Web (WWW '19)*, 2019.

[95] Jacob Abbott and Sameer Patil. How Mandatory Second Factor Affects the Authentication User Experience. In *2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–13, New York, NY, USA, 2020. Association for Computing Machinery.

[96] Mohammed Jubur, Prakash Shrestha, Nitesh Saxena, and Jay Prakash. Bypassing push-based second factor and passwordless authentication with human-indistinguishable notifications. In *2021 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '21, page 447–461, New York, NY, USA, 2021. Association for Computing Machinery.

[97] Anna L. Buczak and Erhan Guven. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. *IEEE Communications Surveys & Tutorials*, 18(2), 2016.

[98] Mahdi Hassan Aysa, Abdullahi Abdu Ibrahim, and Alaa Hamid Mohammed. IoT DDOS Attack Detection Using Machine Learning. In *2020 4th International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*, 2020.

[99] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a "Kneedle" in a Haystack: Detecting Knee Points in System Behavior. In *2011 31st International Conference on Distributed Computing Systems Workshops*, 2011.

[100] Ashutosh Bhardwaj. Silhouette coefficient. https://towardsdatascience.com/silhouette-coefficient-validating-clustering-techniques-e976bb81d10c, 2020.

[101] UA-parser. https://github.com/ua-parser/uap-python.

[102] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[103] Shape Security. Automating cybercrime with sentry MBA: The most commonly used attack tool for credential stuffing. https://www.shapesecurity.com/reports/sentry-mba, 2020.

[104] Pang-Ning Tan, Michael Steinbach, Anuj Karpatneand, and Vipin Kumar. Introduction to Data Mining, 2nd Edition. *People's Posts and Telecommunications Publishing House, Beijing*, 2019.

[105] Wikipedia. List of the most common passwords. https://en.wikipedia.org/wiki/List_of_the_most_common_passwords.

[106] https://blackbox.ipinfo.app.

[107] Mazharul Islam, Sunpreet S. Arora, Rahul Chatterjee, and Ke Coby Wang. Detecting Compromise of Passkey Storage on the Cloud. In *34th USENIX Security Symposium*. USENIX Association, 2025.

[108] E. Dauterman, H. Corrigan-Gibbs, and D. Mazières. SafetyPin: Encrypted backups with Human-Memorable secrets. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 1121–1138, 2020.

[109] M. Zinkus, T. M. Jois, and M. Green. SoK: Cryptographic confidentiality of data on mobile devices. *In Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2022.

[110] A. Juels and R. L. Rivest. Honeywords: Making password-cracking detectable. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 145–160, 2013.

[111] K. C. Wang and M. K. Reiter. Using amnesia to detect credential database breaches. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 839–855, 2021.

[112] A. Dionysiou and E. Athanasopoulos. Lethe: Practical data breach detection with zero persistent secret state. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 223–235. IEEE, 2022.

[113] M. Islam and K. C. Wang. CASPER. `https://archive.softwareheritage.org/browse/origin/directory/?origin_url=https://github.com/islamazhar/CASPER`, 2024.

[114] M. H. Almeshekah, C. N. Gutierrez, M. J. Atallah, and E. H. Spafford. Ersatzpasswords: Ending password cracking and detecting password leakage. In *Proceedings of the 31st Annual Computer Security Applications Conference*, pages 311–320, 2015.

[115] M. Chase, H. Davis, E. Ghosh, and K. Laine. Acsesor: A new framework for auditable custodial secret storage and recovery. *Cryptology ePrint Archive*, 2022.

[116] C. Orsini, A. Scafuro, and T. Verber. How to recover a cryptographic secret from the cloud. *Cryptology ePrint Archive*, 2023.

[117] H. Bojinov, E. Bursztein, X. Boyen, and D. Boneh. Kamouflage: Loss-resistant password management. In *Computer Security–ESORICS 2010: 15th European Symposium on Research in Computer Security, Athens, Greece, September 20-22, 2010. Proceedings 15*, pages 286–302. Springer, 2010.

[118] R. Chatterjee, J. Bonneau, A. Juels, and T. Ristenpart. Cracking-resistant password vaults using natural language encoders. In *2015 IEEE Symposium on Security and Privacy*, pages 481–498. IEEE, 2015.

[119] H. Cheng, W. Li, P. Wang, C.-H. Chu, and K. Liang. Incrementally updateable honey password vaults. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 857–874, 2021.

[120] D. Florêncio, C. Herley, and P. C. Van Oorschot. An administrator's guide to internet password research. In *28th large installation system administration conference (LISA14)*, pages 44–61, 2014.

[121] B. Lu, X. Zhang, Z. Ling, Y. Zhang, and Z. Lin. A measurement study of authentication rate-limiting mechanisms of modern websites. In *Proceedings of the 34th annual computer security applications conference*, pages 89–100, 2018.

[122] Maximilian Golla, Benedict Beuscher, and Markus Dürmuth. On the security of cracking-resistant password vaults. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1230–1241. ACM, 2016.

[123] T. Rao, Y. Su, P. Xu, Y. Zheng, W. Wang, and H. Jin. You reset i attack! a master password guessing attack against honey password vaults. In *European Symposium on Research in Computer Security*, pages 141–161. Springer, 2023.

[124] F. Duan, D. Wang, and C. Jia. A security analysis of honey vaults. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024.

[125] FIDO Alliance. The FIDO ("Fast IDentity Online") Alliance – Industry Association to Promote Authentication Standards. `https://fidoalliance.org/fido2/`, 2024.

[126] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. Technical report, 2008.

[127] Kurt Thomas, Frank Li, Ali Zand, Jacob Barrett, Juri Ranieri, Luca Invernizzi, Yarik Markov, Oxana Comanescu, Vijay Eranti, Angelika Moscicki, et al. Data breaches, phishing, or malware? understanding the risks of stolen credentials. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pages 1421–1434, 2017.

[128] I. Erguler. Achieving flatness: Selecting the honeywords from existing user passwords. *IEEE Transactions on Dependable and Secure Computing*, 13(2):284–295, 2015.

[129] D. Chang, A. Goel, S. Mishra, and S. K. Sanadhya. Generation of secure and reliable honeywords, preventing false detection. *IEEE Transactions on Dependable and Secure Computing*, 16(5):757–769, 2019.

[130] D. Wang, Y. Zou, Q. Dong, Y. Song, and X. Huang. How to attack and generate honeywords. In *43rd IEEE Symposium on Security and Privacy*, pages 966–983. IEEE, May 2022.

[131] K. C. Wang and M. K. Reiter. Bernoulli honeywords. In *31st ISOC Network and Distributed System Security Symposium*, February 2024.

[132] Z. Huang, L. Bauer, and M. K. Reiter. The impact of exposed passwords on honeyword efficacy. *33rd USENIX Security Symposium (USENIX Security 24)*, 2024.

[133] D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ecdsa). *International journal of information security*, 1:36–63, 2001.

[134] A. Juels and T. Ristenpart. Honey encryption: Security beyond the brute-force bound. In *Advances in Cryptology–EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings 33*, pages 293–310. Springer, 2014.

[135] J. Katz, R. Ostrovsky, and M. Yung. Efficient and secure authenticated key exchange using weak passwords. *Journal of the ACM (JACM)*, 57(1):1–39, 2009.

[136] J. Bonneau. The science of guessing: analyzing an anonymized corpus of 70 million passwords. In *2012 IEEE symposium on security and privacy*, pages 538–552. IEEE, 2012.

[137] M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *Computer Aided Verification: 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings 23*, pages 585–591. Springer, 2011.

[138] IBM Security. Cost of a data breach report 2023. `https://www.ibm.com/security/digital-assets/cost-data-breach-report/`, 2023.

[139] Descope. virtualwebauthn. https://github.com/descope/virtualwebauthn, 2024.

[140] F. Amacker. WebAuthn server library (Go/Golang). https://github.com/fxamacker/webauthn, 2024.

[141] J. H. Huh, H. Kim, R. B. Bobba, M. N. Bashir, and K. Beznosov. On the memorability of system-generated PINs: Can chunking help? In *eleventh symposium on usable privacy and security (SOUPS 2015)*, pages 197–209, 2015.

[142] M. Khamis, T. Seitz, L. Mertl, A. Nguyen, M. Schneller, and Z. Li. Passquerade: Improving error correction of text passwords on mobile devices by using graphic filters for password masking. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–8, 2019.

[143] Smartphone providers await 2H smartphone rebound. https://www.bloomberg.com/professional/blog/smartphone-providers-await-2h-smartphone-rebound, Available on: 2024-03-01.

[144] A. Gavazzi, R. Williams, E. Kirda, L. Lu, A. King, A. Davis, and T. Leek. A study of Multi-Factor and Risk-Based authentication availability. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 2043–2060, 2023.

[145] S. Wiefling, M. Dürmuth, and L. Lo Iacono. More than just good passwords? a study on usability and security perceptions of risk-based authentication. In *Proceedings of the 36th Annual Computer Security Applications Conference*, ACSAC '20, page 203–218, New York, NY, USA, 2020. Association for Computing Machinery.

[146] S. Wiefling, L. Lo Iacono, and M. Dürmuth. Is this really you? an empirical study on risk-based authentication applied in the wild. In *ICT Systems Security and Privacy Protection: 34th IFIP TC 11 International Conference, SEC 2019, Lisbon, Portugal, June 25-27, 2019, Proceedings 34*, pages 134–148. Springer, 2019.

[147] M. Khonji, Y. Iraqi, and A. Jones. Phishing detection: a literature survey. *IEEE Communications Surveys & Tutorials*, 15(4):2091–2121, 2013.

[148] M. Workman. Wisecrackers: A theory-grounded investigation of phishing and pretext social engineering threats to information security. *Journal of the American society for information science and technology*, 59(4):662–674, 2008.

[149] P. Markert, A. Adhikari, and S. Das. A transcontinental analysis of account remediation protocols of popular websites. *arXiv preprint arXiv:2302.01401*, 2023.

[150] Leland McInnes, John Healy, and Steve Astels. HDBSCAN: Hierarchical Density-Based Clustering. *The Journal of Open Source Software*, 2(11), 2017.

[151] Aleksandar Lazarevic, Levent Ertoz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of the 2003 SIAM international conference on data mining*, pages 25–36. SIAM, 2003.

[152] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[153] OpenBullet. `https://openbullet.github.io/`, 2022.

[154] C. Gilsenan, F. Shakir, N. Alomar, and S. Egelman. Security and privacy failures in popular 2FA apps. In *32nd USENIX Security Symposium (USENIX Security 23)*, 2023.

[155] Experts Fear Crooks are Cracking Keys Stolen in LastPass Breach. `https://krebsonsecurity.com/2023/09/experts-fear-crooks-are-cracking-keys-stolen-in-lastpass-breach/`, Available on: 2024-03-01.

[156] LastPass' latest data breach exposed some customer information. `https://www.theverge.com/2022/11/30/23486902/lastpass-hackers-customer-information-breach`, Available on: 2024-03-01.

[157] Apple. Escrow security for iCloud Keychain. `https://support.apple.com/guide/security/escrow-security-for-icloud-keychain-sec3e341e75d/1/web/1`, 2024.

[158] A. Weinert. How it works: Backup and restore for Microsoft Authenticator. https://techcommunity.microsoft.com/t5/microsoft-entra-blog/how-it-works-backup-and-restore-for-microsoft-authenticator/ba-p/1006678, 2019.

[159] A. Birgisson. Security of Passkeys in the Google Password Manager. 2023. https://security.googleblog.com/2022/10/SecurityofPasskeysintheGooglePasswordManager.html.

[160] National Vulnerability Database. CVE-2023-42847. https://nvd.nist.gov/vuln/detail/CVE-2023-42847, Published on: 2023-10-25.

[161] Apple Plaftorm Security, iCloud Keychain security overview. https://support.apple.com/guide/security/icloud-keychain-security-overview-sec1c89c6f3b/1/web/1, Available on: 2025-01-30.

[162] L. Lassak, P. Markert, M. Golla, E. Stobert, and M. Dürmuth. A comparative long-term study of fallback authentication schemes. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, 2024.

[163] H. Krawczyk. Cryptographic extraction and key derivation: The HKDF scheme. In *Annual Cryptology Conference*, pages 631–648. Springer, 2010.

[164] Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parameters. https://www.secg.org/sec2-v2.pdf, 2010.

[165] L. Ballard, S. Kamara, F. Monrose, and M. K. Reiter. Towards practical biometric key generation with randomized biometric templates. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 235–244, 2008.

[166] D. M'Raihi, M. Bellare, F. Hoornaert, D. Naccache, and O. Ranen. RFC 4226: HOTP: An HMAC-based one-time password algorithm, 2005.

[167] D. M'Raihi, S. Machani, M. Pei, and J. Rydell. RFC 6238: TOTP: Time-based one-time password algorithm, 2011.