Gossamer: Securely Measuring Password-based Logins

Marina Sanusi Bohuk
Cornell University

Mazharul Islam
University of Wisconsin–Madison

Suleman Ahmad
Cloudflare[*]

Michael Swift
University of Wisconsin-Madison

Thomas Ristenpart
Cornell Tech

Rahul Chatterjee
University of Wisconsin-Madison

Abstract

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.

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. [10] 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, such as the similarity between the successive password submissions. Chatterjee et al. [11] 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 [10] or compared the hashes of a small handful of variants of a submitted password to the real user's password hash [11]. 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

[*] Work done while at UW–Madison

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 [31] (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 [47]) 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 [35]. 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, our paper is the first to propose 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.

2 Background and Related Work

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 [8] is a common way to achieve SSO.

Looking ahead 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 4).

Studies about passwords. Prior works [20, 26, 30, 32, 46] 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 [22, 24, 43], under different factors such as password requirements [24], presence of a password strength meter [43], and use of a password blocklist [18].

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 [14, 15, 36]. On the client side, Florencio and Herley [14] installed a Windows Live Toolbar component for five hundred thousand volunteer participants and analyzed their password behavior over 85 days. Similarly, Forget et al. [15] created a client-side data collection tool to observe user's password behavior in its natural environment [15, 36].

Measurement studies with login systems. To our knowledge, three studies have looked at user passwords by instrumenting the login servers. Bonneau et al. [10] 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. [29] correlated password strength with demographic information in an offline study with reversibly encrypted passwords on an access-restricted computer. Chatterjee et al. [11] 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 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.

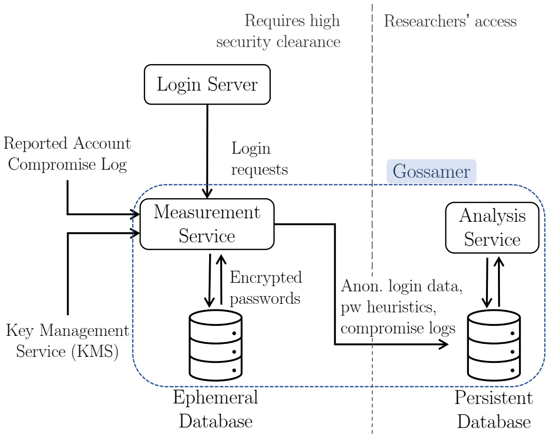

Figure 1: The main components of Gossamer.

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 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] of the HTTP headers, timestamp, login result (success 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

[1] The login server removes sensitive cookies specified by the security engineers from the HTTP header, as some content can be more sensitive than user passwords. For example, an "authentication cookie" could bypass MFA requirements.

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 [31] 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 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 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 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 [19]. 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 [33, 41] 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 [4] library to encrypt user passwords in the ephemeral database using AES-256, and we use the Python Miscreant [5] 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.

[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.

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.

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.

4.1 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 10 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 [47], and whether or not it belongs to one of the popular password guessing lists (the most frequent 5,000 passwords in RockYou [6] or the top 5,000 passwords generated[3] by Hashcat [40]).

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 [45] and the Compilation of Many Breaches (COMB) containing 3.2 billion pairs [37] 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 [12, 34, 44] that target passwords similar to a user's other breached passwords. We therefore consider recording the submitted password's edit distance, PPSM similarity [34], and Pass2Path similarity [34] 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.

4.2 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 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

[3] We use the rule list `best_64.rule` [7] (a rule list compiled by the Hashcat community of what are considered to be the best 64 rules) with RockYou to generate the guesses.

[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.

appears in data breaches known to the attacker, and so the best strategy is to modify a targeted password guessing attack (e.g., [34]) 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 [25, 34] 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 [25, 34]. 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 [47] 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 $w \in \mathsf{W}$ 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 2. We first discuss the online context, where $q \leq 1000$. 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).

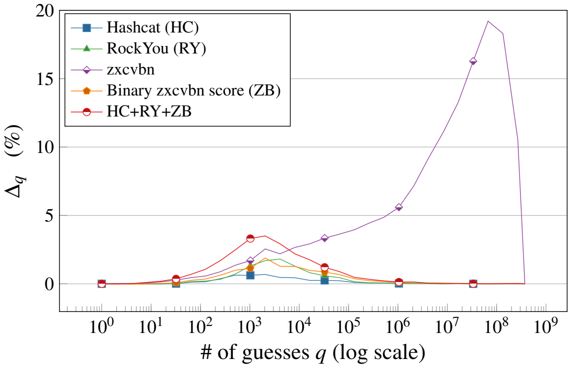

Figure 2: Relative improvement in password guessing success rate (Δ_q) due to access to password-derived statistics from Gossamer over the baseline in $q \leq 10^8$ guesses.

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 [34], 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 ≤ 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 Figure 10.

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, and any summary statistic that may be affected an update was calculated using the data after the update was made .

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

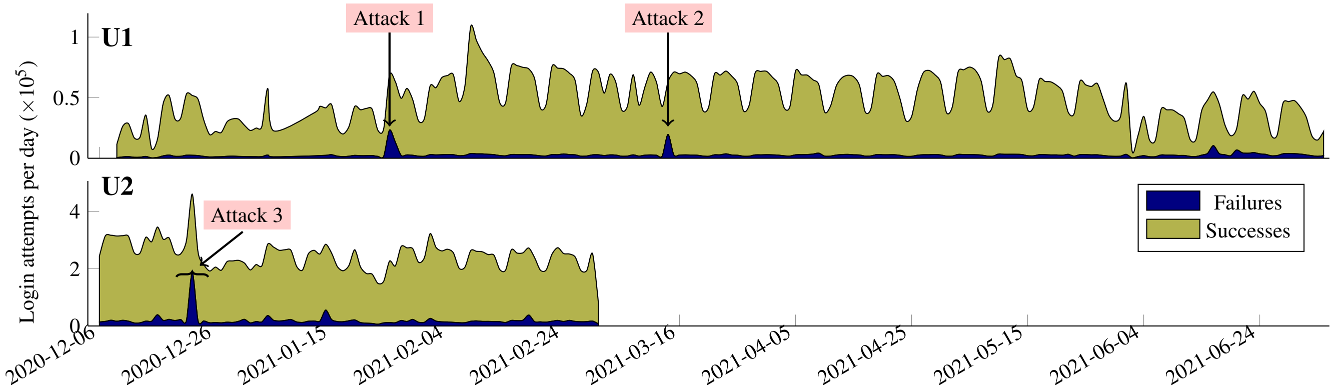

Figure 3: Successful and failed login attempts per day at two universities (for a total of 196 K unique users at U1 and 309 K at U2). Potential high-volume attack campaigns we discovered are also shown.

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 benefits 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., [21]) and are discussed as an acceptable approach in the Menlo report [23].

Deployment configuration. As mentioned in Section 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 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 [13] 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.

6 Login Statistics, Patterns, & 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 3. 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

	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
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%

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

Figure 4: Summary statistics of login requests recorded by Gossamer at U1 and U2. More statistics can be found in Appendix A Figure 12.

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 [27] 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 Figure 4, with additional statistics in Figure 12 in Appendix A.

6.1 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 [28] as coming from NordVPN, one from Microsoft, and one from Inwi Mobile [3]. 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 [2] 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 [42]. 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 [1] 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.

6.2 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 Figure 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 Figure 13. In 22% of requests at U2, the user agent string was set to an empty string; all of these requests were through basic authentication [16]. 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 11.

Prior work [17] 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 [17]. 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 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 device (smartphone or tablet) 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. At U2, 18% of requests (80% of which were iOS and 20% Android) came from a mobile device. At both schools, iOS devices were significantly more popular.

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.

6.3 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 se-

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

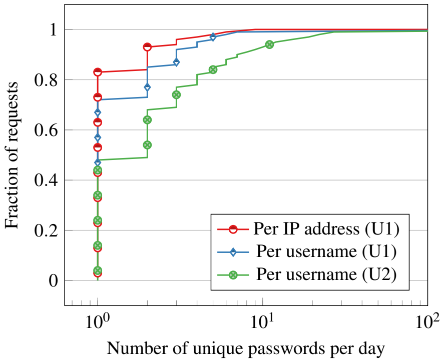

Figure 5: Cumulative distributions of unique passwords per username and IP for February 2021. The X-axis is log scale.

curity 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 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 [7]; and finally, whether the password appears in the top 1000 most common passwords in our breach compilation dataset.

Exact percentages of requests matching each of these password strength metrics can be found in Appendix A Figure 12. 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 at U1 (unfortunately we do not have this measurement at U2 because of a configuration issue) . Both distributions are shown in Figure 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 [34]. 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. [34] 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 [35] or using a personalized password strength meter [34] 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. Although we did not instrument the password change system, 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 — 78 of which appeared in the compromise database — 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 [10, 14].

6.4 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 [11, 38, 39]. 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 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*.

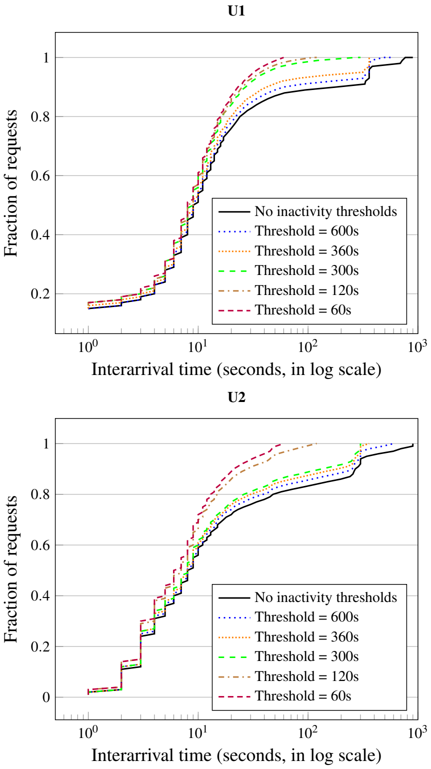

Figure 6: Cumulative distribution of interarrival time between requests for different inactivity thresholds for February 2021. The X-axis is log scale.

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 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 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 (Figure 7) 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

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

Figure 7: Average session sizes (number of attempts per session) for different inactivity thresholds

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 Figure 7. At U1, we found that 22% of successful sessions and 14% of abandoned sessions required more than one attempt; at U2, 38% of sessions required more than one attempt. 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 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 8. 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 9. 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 submission was within edit distance two of the actual password.

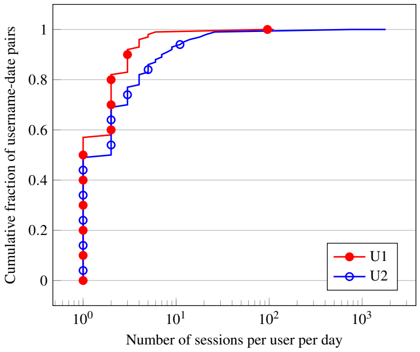

Figure 8: Cumulative distributions of number of sessions per user per day for an inactivity threshold of 360s for February 2021. The X-axis is log scale.

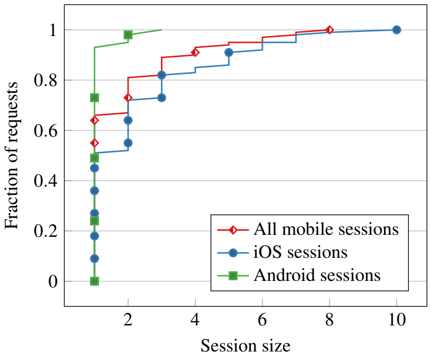

Figure 9: Cumulative distributions of session sizes (number of attempts per session) for mobile devices for February 2021. The X-axis for U2 is limited to 10 for comparison with U1 (although the maximum session size consisted of 54 attempts).

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. [11] 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. Since, as we show in Figure 4, the majority of users — 99.97% users in U1 and 99.94% users in U2 — are using strong passwords (according to the zxcvbn score), we believe that these users are using password managers to enter their passwords.

Duo logs. Both U1 and U2 have introduced the use of Duo two factor authentication [9] 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 [16], 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.

7 Conclusion

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.

Through these measurements, we observed several patterns in user login behavior. First, we saw that login friction is still high. Many users require more than one attempt before successfully logging in, and Duo two-factor authentication added on average an additional 14 seconds to their login. One suggestion we have for reducing login friction is to count unique passwords submitted for a user, rather than every request; we find that this would reduce lockouts by 88%.

Second, we observe that reuse of breached credentials is a serious problem: nearly 3% of valid users at U1 and more than 9% at U2 are using a breached password that appears in our breach dataset. During our data collection period, some users changed their passwords to either exactly equal to one of their breached passwords or very similar to their breached passwords. To reduce this number, we encourage the deployment of breach alerting services such as Might I Get Pwned [35] that could indicate when a user's password is vulnerable to credential stuffing.

We also find that visibility into submitted passwords is helpful for designing security mechanisms and informing new policies. Similar to how we proposed a new lockout policy, authentication system developers can use a framework such as Gossamer to test the effect of new policies before their deployment. Future analyses with Gossamer may investigate how password-derived signals help in detecting different types of password guessing attacks and the efficacy of Duo two-factor authentication in stopping different types of attacks. Also, Gossamer can be extended with additional measurements using the same framework and simulation method for assessing the risk of different measurements. Thus both the tool and approach can be useful in the future for helping analysts detect and develop mitigations against attacks. We will be releasing Gossamer as a public, open-source project.

Acknowledgments

We thank the IT engineers, especially Dan Villanti, Jerry Shipman, Ryan Larschedit, and Bruce LaBuda, and the IT leadership at both universities for their support in the research and helping us set up the instrumentation. We thank the anonymous reviewers for their insightful feedback, which helped improve the paper significantly. This work was is supported in part by the University of Wisconsin–Madison Office of the Vice Chancellor for Research and Graduate Education with funding from the Wisconsin Alumni Research Foundation and NSF grant CNS #1763810.

References

[1] DigitalOcean – the developer cloud. URL: https://www.digitalocean.com/ [cited May 11, 2022].

[2] Google Cloud: Cloud Computing Services. URL: https://cloud.google.com/ [cited May 11, 2022].

[3] Inwi – opérateur téléphonique mobile, fixe & internet au maroc. URL: https://www.inwi.ma/ [cited May 11, 2022].

[4] Python fernet cryptography library. URL: https://cryptography.io/en/latest/fernet.html [cited May 11, 2022].

[5] Python miscreant encryption library. URL: https://github.com/miscreant/miscreant.py [cited May 11, 2022].

[6] RockYou hack: From bad to worse, 2009. URL: https://techcrunch.com/2009/12/14/rockyou-hack-security-myspace-facebook-passwords/ [cited May 11, 2022].

[7] Best64 rule list, 2018. URL: https://github.com/hashcat/hashcat/blob/master/rules/best64.rule [cited May 11, 2022].

[8] OAuth, 2020. URL: https://oauth.net/2/ [cited May 11, 2022].

[9] Duo Security: Two-Factor Authentication & End Point Security, 2021. URL: https://duo.com/ [cited May 11, 2022].

[10] 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. doi:10.1109/SP.2012.49.

[11] 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.

[12] 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.

[13] 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.

[14] 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, page 657–666, New York, NY, USA, 2007. Association for Computing Machinery. URL: https://doi.org/10.1145/1242572.1242661, doi:10.1145/1242572.1242661.

[15] 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. URL: https://doi.org/10.1145/2600176.2600200, doi:10.1145/2600176.2600200.

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

[17] D Freeman, Sakshi Jain, Markus Duermuth, Battista Biggio, and Giorgio Giacinto. Who are you? a statistical approach to measuring user authenticity. In *NDSS*, 2016. doi:10.14722/ndss.2016.23240.

[18] 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.

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

[20] Briland Hitaj, Paolo Gasti, Giuseppe Ateniese, and Fernando Perez-Cruz. PassGAN: A Deep Learning Approach for Password Guessing, 2019. arXiv:1709.00440.

[21] 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.

[22] 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. doi:10.1109/SP.2012.38.

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

[24] 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. URL: https://doi.org/10.1145/1978942.1979321, doi:10.1145/1978942.1979321.

[25] 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. URL: https://doi.org/10.1145/3319535.3354229, doi:10.1145/3319535.3354229.

[26] J. Ma, W. Yang, M. Luo, and N. Li. A study of probabilistic password models. In *2014 IEEE Symposium on Security and Privacy*, pages 689–704, 2014. doi:10.1109/SP.2014.50.

[27] MaxMind. GeoIP2 Database. URL: https://www.maxmind.com/en/geoip2-databases [cited May 11, 2022].

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

[29] 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 *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, page 173–186, New York, NY, USA, 2013. Association for Computing Machinery. URL: https://doi.org/10.1145/2508859.2516726, doi:10.1145/2508859.2516726.

[30] 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. URL: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/melicher.

[31] 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. URL: https://www.microsoft.com/en-us/research/publication/bunker-a-privacy-oriented-platform-for-network-tracing/.

[32] Arvind Narayanan and Vitaly Shmatikov. Fast dictionary attacks on passwords using time-space tradeoff. pages 364–372, 2005. doi:10.1145/1102120.1102168.

[33] 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.

[34] B. Pal, T. Daniel, R. Chatterjee, and T. Ristenpart. Beyond credential stuffing: Password similarity models using neural networks. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 417–434, 2019. doi:10.1109/SP.2019.00056.

[35] Bijeeta Pal, Mazharul Islam, Thomas Ristenpart, and Rahul Chatterjee. Might I get pwned: A second generation password breach alerting service, 2021. arXiv:2109.14490.

[36] 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. URL: https://doi.org/10.1145/3133956.3133973, doi:10.1145/3133956.3133973.

[37] Tara Seals. Billions of passwords offered for $2 in cyberunderground. *ThreatPost*, 2021. URL: https://threatpost.com/billions-passwords-cyber-underground/163738/.

[38] 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. URL: https://doi-org.proxy.library.cornell.edu/10.1145/2335356.2335366, doi:10.1145/2335356.2335366.

[39] 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. URL: https://doi-org.proxy.library.cornell.edu/10.1145/2556288.2557377, doi:10.1145/2556288.2557377.

[40] Jens Steube and Gabriele Gristina. Hashcat. URL: https://hashcat.net/hashcat/ [cited May 11, 2022].

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

[42] Danna Thee. Sentry MBA. URL: https://e.cyberint.com/hubfs/Sentry%20MBA%20Report/CyberInt_Sentry-MBA_Report.pdf [cited May 11, 2022].

[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] Ding Wang, Zijian Zhang, Ping Wang, Jeff Yan, and Xinyi Huang. Targeted online password guessing: An underestimated threat. 2016. doi:10.1145/2976749.2978339.

[45] WAQAS. A trove of 1.4 billion clear text credentials file found on dark web, 2017. URL: https://www.hackread.com/billion-of-credentials-found-on-dark-web/ [cited May 11, 2022].

[46] M. Weir, S. Aggarwal, B. d. Medeiros, and B. Glodek. Password cracking using probabilistic context-free grammars. In *2009 30th IEEE Symposium on Security and Privacy*, pages 391–405, 2009. URL: https://web.eecs.utk.edu/~mschucha/netsec/readings/cfgPass.pdf, doi:10.1109/SP.2009.8.

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

A Appendix

A.1 Measurements Taken

We show all data stored in the ephemeral and persistent databases in Figure 10. Note that the raw password is only stored in encrypted format for 24 hours in the ephemeral database. The symbol × indicates that a field was stored in plain text and ⊗ 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. 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 paper, 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

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 [34] strength of password		×
guess rank due to credential tweaking attack [34]		×
edit dist. ≤ 2 of pw from		
– other submissions for same username		×
– other submissions for same IP		×

Figure 10: Measurements we log in ephemeral (**Eph.**) and persistent (**Pers.**) storage.

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. Figure 12 shows some additional statistics we reported earlier in Figure 4. Figure 11 shows the distribution of operation systems as parsed from the user agents of the requests at both universities; Figure 13 shows the top 10 most common user agents we saw at both schools.

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%

Figure 11: 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
– 99th-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.

Figure 12: 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%

Figure 13: Top 10 most common user agents at U1 (**top**) and at U2 (**bottom**)