Authentication Defense Simulator: Measuring Security vs Usability for Basic Login Defenses

Author: Varun Umapathy

Date: 01/03/2026

Abstract

Password logins are typically a favorite of attackers, who may attempt brute-force guessing or use "botnets" with low-rate attempts. While these defenses (lockouts, rate limiting, and backoff) can certainly enhance system security, they can also negatively impact system usability by locking out valid users who enter the password incorrectly or attempt to log in too rapidly. In this project, I created a Python-based simulator to illustrate how various types of attackers and legitimate users access a login system with different levels of defensive measures. I examined four defense mechanisms: Lockout, Rate Limiting Per Account, Rate Limiting Per IP, and Exponential Backoff. I assessed both security (whether a targeted account could be compromised) and usability (how frequently legitimate users were locked out, and how many unique users experienced at least one block). Using a 1-hour simulation per trial (3 seeds per setting), Lockout effectively prevented compromise of the targeted accounts, while no user experienced a lockout during this trial. Similarly, backoff effectively prevented compromise of the targeted accounts but resulted in measurable user locks and impacted many unique users during the trial period. In the configuration used in this research, neither account-based nor IP-based rate limiting provided sufficient protection against compromise of targeted accounts before the end of the simulated trial period.

1. Introduction

Still, the majority of users employ passwords for login credentials for their online services, and while the ease of using passwords is beneficial, it is also a drawback for users; attackers can attempt to guess a user's password until they successfully guess the password, or until the system no longer allows additional failed login attempts. As with many attacks, attackers may have access to shared password lists or leaked credential pairs (username/password) that were previously compromised via another website, as well as automated tools that may rapidly submit possible username/password combinations. Although the attacker may be guessing the user's password at a slow rate, with enough patience, they may eventually guess it correctly, especially if there is little to no enforcement limiting repeated unsuccessful login attempts. The general guidance for defending against authentication-related threats, such as throttling and lockout-type protections, as well as protecting accounts, is outlined in both OWASP and NIST publications [1][2]. OWASP also describes Credential Stuffing as a threat to systems [3].

As a result, most systems include some form of protection. While some protections can be strict (for example, locking out a user’s account after a few unsuccessful attempts), other protections will hinder the attacker's ability to make rapid attacks, while still allowing legitimate users to

access the system (i.e., rate limiting). However, a problem with these protections is that they affect both legitimate users and attackers. For example, if a user enters their password incorrectly a couple of times and then has their account locked out by the system, they are prevented from accessing the system until the lockout expires or support resets it. In practice, frustrated users call customer support, increasing support costs. As such, there is a classic trade-off between usability and security in real-world systems.

This project investigates a practical question: if we simulate a real login system and create reasonable models of attacker and user behavior, what will happen when we test various levels of defense?

I am not attempting to create a perfect representation of the Internet. Still, the simulation approach has value as it provides a repeatable process for testing and allows us to identify causal relationships between variables. Instead of hypothesizing which defense "should" provide the most significant benefit, I can design and conduct controlled experiments to determine the relative effectiveness of the defenses.

The purpose of this report is to explain the system that I developed, describe my experiment methodology, and present data and results related to the four defense mechanisms that were implemented. I used a research-based format to write this report so that it would look and feel like a typical computer science experiment: i.e., what was investigated, how it was investigated, and what the results were.

Repository: https://github.com/VarunUmapathy07/authentication-defense-simulator
Setup and experiment instructions are provided in the repository README.

2. Related work

Guessing and credential-stuffing attacks are still widely used against online sites. Lockout, rate limits, and back-off timers are common responses by sites to defend against such attacks. However, these same defense mechanisms can deny access to legitimate users. This denial becomes more likely as more users share a public IP address (e.g., a university dorm or college campus network).

2.1 What websites actually do

Lu et al. demonstrated [5] the large variability in login throttling across real-world sites, where the same rate-limiting rule affects both the likelihood that an attacker successfully compromises an account and the likelihood that a normal user is locked out of their account. In addition, lockouts have introduced another problem: attackers can intentionally trigger lockouts and then deny legitimate users access to their accounts, effectively turning a defensive mechanism into a

denial-of-service (DoS) mechanism [6]. Therefore, when evaluating a defense mechanism for its effectiveness against compromise attempts, one needs to consider its impact on legitimate users.

2.2 Where simple counting breaks down

Many standard defense systems will take into account the number of incorrect login attempts and apply punishment to either the account itself or the source IP address for those failed login attempts. Account-based throttling is most effective when dealing with one attacking source IP address. However, if the attacking botnet uses multiple source IP addresses to guess passwords for the same target account, it can try far more passwords in the same time window. IP-based throttling may not flag any single IP because each one stays under the threshold. This creates a disconnect between account-based throttling systems (which focus on the account) versus IP-based throttling systems (which focus on the source IP).

2.3 Going beyond lockout and basic throttling

While some work has attempted to improve online guessing defenses with methods beyond just the use of a fail counter, some systems have found ways to strengthen their ability to defend against guessing in real-time based on signals that are used to identify potential attacks (for example, if an attacker is submitting a password that appears on common list of guessed passwords), therefore increasing the strength of the system's response to these types of attempts [7]. While some defenses limit the number of attempts an attacker can make before locking out the account, other defenses may allow users to continue attempting to log in until they reach a certain threshold or until a large amount of time has passed [8]. The fact that a system can look very effective from a theoretical standpoint does not mean it will be effective for all users, nor that it will not cause problems for users when they are locked out due to excessive login attempts. Therefore, you need a method that clearly defines how your policy compares to others for the same workload.

2.4 Credential stuffing context

Credential stuffing works because people reuse their passwords and use similar patterns across sites. Research studies have shown that reuse is frequent enough to be effective in this manner [9]. Additional research has focused on ways to prevent credential stuffing by issuing warnings to both users and providers when they are identified in breach data, enabling a quicker account reset for the user [10]. Another area of research involves detection methods that identify when an attacker attempts to stuff credentials across a user's multiple accounts [11]. This research focuses on live detection and response. This project compares how well login gates perform with different defender actions relative to attacker behavior. It illustrates the trade-off between security and usability.

2.5 Where this project fits

Real-world measurements are presented by measurement papers [5], while other research projects suggest new approaches to defend against attacks [7], [8]. The goal of your project is to create a local simulation environment that can be run multiple times, supports parameter sweeps, and allows you to compare different defense strategies on the same workload, while measuring both user-blocking and compromises. This work focuses on controlled comparisons under identical workloads, including shared-IP effects and distributed low-rate attackers. It reports a security-versus-usability frontier so you can see how each policy trades off compromise prevention against user blocking.

3. Problem Statement and Goals

The problem lies in repeated authentication attempts by an attacker on a user's account. The attacker will continue to try various passwords until they successfully obtain access to the account, particularly if the system does not limit the number of failed login attempts. This type of attack is likely to occur when a user's password appears on a standard password list. There are a variety of systems that may provide some level of protection against this type of attack; however, the effectiveness of those protections varies depending on whether the attack is fast or distributed and whether legitimate users will experience negative impacts from their own mistaken login attempts.

There are four main goals of this research. To achieve my first objective of developing a simulation-based modeling framework that represents the behavior of attackers and legitimate users over time, I need to create a model that captures both types of behavior. My second objective is to design and implement several different defense policies and apply them consistently so that they can be evaluated relatively. To assess the success of the defenses, I need to use both security- and usability-based measurements to determine whether the defense is effective, i.e., whether it prevents attackers from accessing the user's account without adversely affecting legitimate users who incorrectly enter their password. My fourth objective is to report the results clearly (in the form of charts and/or tables) and, based on those results, to make recommendations on when each defense policy should be applied.

4. Overview of the Simulator

My Python event-based simulation works as follows: Actors (both attackers and regular users) will not perform actions continuously. Instead, each actor has a "next time" when it will try to log

in to the application. Each actor's next event is stored in a queue, along with its time of occurrence. At each time step, the simulation advances its clock to the next event in the queue, performs the action represented by that event, records the details of what occurred, and schedules the actor's next event. A simulation is often implemented this way because it is efficient. By doing so, it does not require stepping through each second of the day when no action is occurring.

There are four primary components of this system. The first component is the actors' behavior. In general, the attacker generates login attempts based on a list of possible password candidates. For regular users, the generated login attempts follow a user-specified schedule and may also include typos or retry attempts. The second component is the authentication service. When a login attempt is made, the service checks the credentials submitted by the user and, before deciding whether the effort is approved, denied, or delayed, invokes a defense policy. The third component is the defense policy layer. In this layer, a specific rule is applied to the request (such as locking out the account or applying rate limits). The final component is logging and analysis. The raw event stream is converted to human-readable metrics for further study.

While this project was created to be easy to understand and reproducible, it is not intended to be production-ready. All the data structures used are fundamental, all the outputs are plain-text files, and some scripts can be run from the command line. The output format is designed to allow you to rerun the same code multiple times to compare results.

5. Threat Model (Attackers)

The simulated attacks attempt to obtain unauthorized access to an account named "victim". The attacker model is as simple as possible; however, it captures two critical features commonly observed among real attackers.

One attacker is a brute-force attacker, using only one IP address. This attacker guesses passwords at a reasonable rate and goes through a list of passwords. In the simulation, the password list contains common passwords and the correct password for the victim account. This was done on purpose, so that a compromise will occur if the defenses do not prevent the attacker in time. If the correct password were not included in the password list, the attacker could never successfully compromise the victim's account, and therefore, the experiment would provide no useful information.

The second attacker model simulates a distributed low-rate attack, which represents a botnet. Instead of having a single attacker make many requests, there are multiple attackers who each make a few attempts (at a slow rate), and each attacker uses a different IP address. This type of attack can evade those defenses that monitor only IP-based request rates, since each IP makes

only normal-looking requests. However, from the targeted account's perspective, the total number of attempts may still be significant.

This threat model was created solely to model common attack types and is not intended to capture all potential sophisticated attacker strategies. For example, this threat model does not include an attacker capable of adaptively altering their behavior in response to defensive feedback. This threat model aims to enable the comparison of various defensive strategies in a controlled environment and to demonstrate how different strategies respond to common threat types.

6. User Model (Normal Users)

Users are "normal" if they attempt to log in to the system as expected. Each normal user will have a Username, Password, and IP Address. Regular users will log in to the simulated environment at times determined by their schedule. Once logged in, the model schedules the user's next login. A normal user may make a mistake entering a username/password combination. If so, the normal user will enter the wrong username/password combination again. The normal user will also be blocked from logging into the system when the Defense is actively blocking the user. When a normal user is blocked from logging into the system, they will simply wait and attempt to log back in later.

An important factor in creating realistic simulations is that most people do not use unique IP Addresses. Often, multiple people share a single IP Address due to Network Address Translation (NAT) (e.g., a home router, a university network, an office network). If a Defense Policy uses IP Addresses to determine which users should be allowed/denied access to the system, it could punish many legitimate users because of one individual's actions on the same network. Therefore, the simulation includes a Shared-IP Group to illustrate this issue.

The User Model has been made as basic as possible, while still providing enough functionality to support error messages, retries, and modeling real-world interactions between users and defenses. The goal of the User Model was to evaluate the usability impact of various defense policies (in addition to the baseline) rather than to provide an exact simulation of user behavior.

7. Defense Policies (What I Tested)

The defense policies tested in this project are all triggered by a user's attempt to log in. Once the defense policy has determined that the attempted login is to be allowed, denied, or delayed, it may also choose to take a specific action (such as locking an account or adding time before the next login).

Lockout will block a user from logging into their account after a certain number of failed attempts to guess the username and password, for a specified period of time. Lockouts are very effective at preventing brute-force attacks on passwords because they limit how quickly a hacker can guess a username and password. However, lockouts can restrict the usability of a system and be abused by hackers to create a denial-of-service attack against a legitimate user account. In this simulation, lockout was used as a direct defense against repeated attempts to guess usernames and passwords.

Rate Limiting Per Account limits the number of login attempts for a particular user account over a specified time frame. This type of limitation prevents guessing of user names and passwords for a particular user account, regardless of the IP Address from which login attempts originate. However, it does guarantee no compromise, since an attacker with sufficient time could still guess the correct username and password.

Rate Limiting Per IP will limit the number of login attempts that can come from a single IP Address over a specified time frame. A single attacking computer could be stopped using this method; however, it would likely not be as effective against a botnet attack, since each attacking computer would use a different IP Address. Also, this method could harm shared networks where many legitimate users appear to be coming from the same IP Address.

Exponential Backoff will increase the delay between each of a user's failed login attempts. Exponential Backoff is like rate limiting, but instead of punishing the user after one failed login attempt, the longer a user continues to fail to guess the username and password, the more severe the punishment becomes for each subsequent failed attempt. This method could prevent repeated guessing since each failed login attempt will cost the attacker more time and effort. However, this method could also frustrate legitimate users who mistyped their username and/or password or have tried to log in multiple times without successfully guessing the correct one.

Hybrid Policies will enforce multiple rules to check for compliance (implemented, but not included in the sweep plots in this version). For example, Hybrid Policies may include both account-based rules and IP Address-based rules. In theory, Hybrid Policies will provide better protection than any single rule, since they will not rely on any single indicator of potential hacking. In practice, the effectiveness of Hybrid Policies will depend on how well they are designed and tuned.

8. Experimental Method

To assess the performance of each defense mechanism, I repeat the simulations multiple times and compile statistical summaries of the results using the simulated event logs. I don't assign a

"default" value for each defense mechanism; instead, I conduct a parameter sweep to demonstrate how the security/usability trade-off varies as the parameter is changed.

The specific configuration used to create each defense is run in triplicate with different random seeds to simulate randomness, and each run is simulated for 1 hour (3600 seconds). A victim account is created, and regular users and attackers are generated. The event loop is then executed, and all successful and failed authentication events are logged. The normal users are represented by 50 users (shared IP allowed,) while the victim represents a single account.

To clearly illustrate the impact of usability in the frontier plot, I utilize a stressed legitimate workload: normal users have a higher typo probability when attempting to authenticate for the first time; normal users also try to authenticate more frequently; and a larger number of users are assigned the same IP address (simulating a shared network), therefore if a defense mechanism throttles or denies attempts it can be easily observed blocking a portion of the legitimate traffic, whereas under a low workload scenario the usability costs would be extremely close to zero.

One parameter is swept per defense mechanism. The number of failures before lockout is limited to 3, 5, or 10, and the duration of the lockout period is 300 seconds. The total number of failures is capped at the specified limit. The base delay of backoffs is swept over 0.25, 0.5, 1.0, and 2.0 seconds (but the maximum delay is capped at 60 seconds). The token bucket configurations used to represent both account-based and IP-based rate limits are swept over a small number of predefined combinations of (max_tokens, refill_rate).

Once the sweep is complete, I convert the per-trial logs into two output files: a file (summary.csv) which contains a summary of the data for each trial, and a file (summary_aggregated.csv) which provides an aggregated view of the data for all of the trials (mean and standard deviation for each configuration). Finally, I produce a security-usability frontier plot using the aggregated data. Each point is the average across the three seeds.

9. Metrics

In my sweep pipeline, compromise_rate is the proportion of an attacker's login attempts that succeed in authenticating to the victim account (the number of successful attacker attempts against the victim username divided by the total number of attempts made by an attacker). time_to_compromise in this context represents the timestamp at which the first successful attacker logs in to the victim; if the victim was never compromised during the simulation, time_to_compromise will equal the duration of the trial (i.e., 3600 seconds). Throughput is the

total number of login attempts per second over the simulation (total log events divided by trial length). For each configuration, I run three seeds and report the mean ± standard deviation across seeds.

I report two usability metrics. One metric is the user block rate, which is the percentage of normal-user login attempts that were blocked by the defense. The other usability metric is impacted_users_pct, which is the percentage of unique users that were blocked at least once by the defense. As such, these two usability metrics represent different ways of viewing usability. A defense may have a lower block rate; however, it could negatively affect many unique users. Another defense may negatively affect fewer users; however, the adverse effect on those users is repeated.

10. Results

This section presents the results of the parameter sweep and highlights the trade-offs between usability and security for each defense's configuration. In this plot, each data point represents one of the configurations that were run (one setting per parameter), and each point is the average across the three seeds.

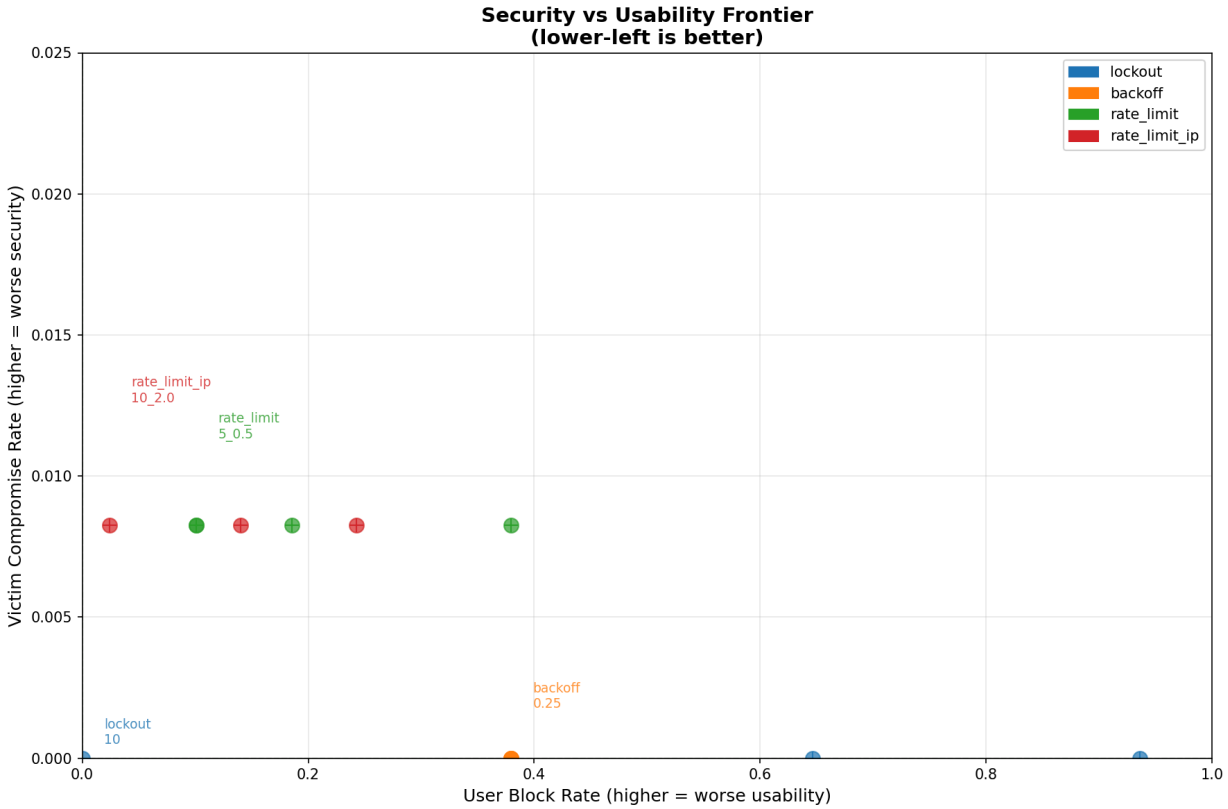

Figure 1. Security vs usability frontier (baseline attacker). Each point is one defense setting. X = user block rate (higher = worse usability). Y = victim compromise rate (higher = worse security). Points show the mean across 3 seeds.

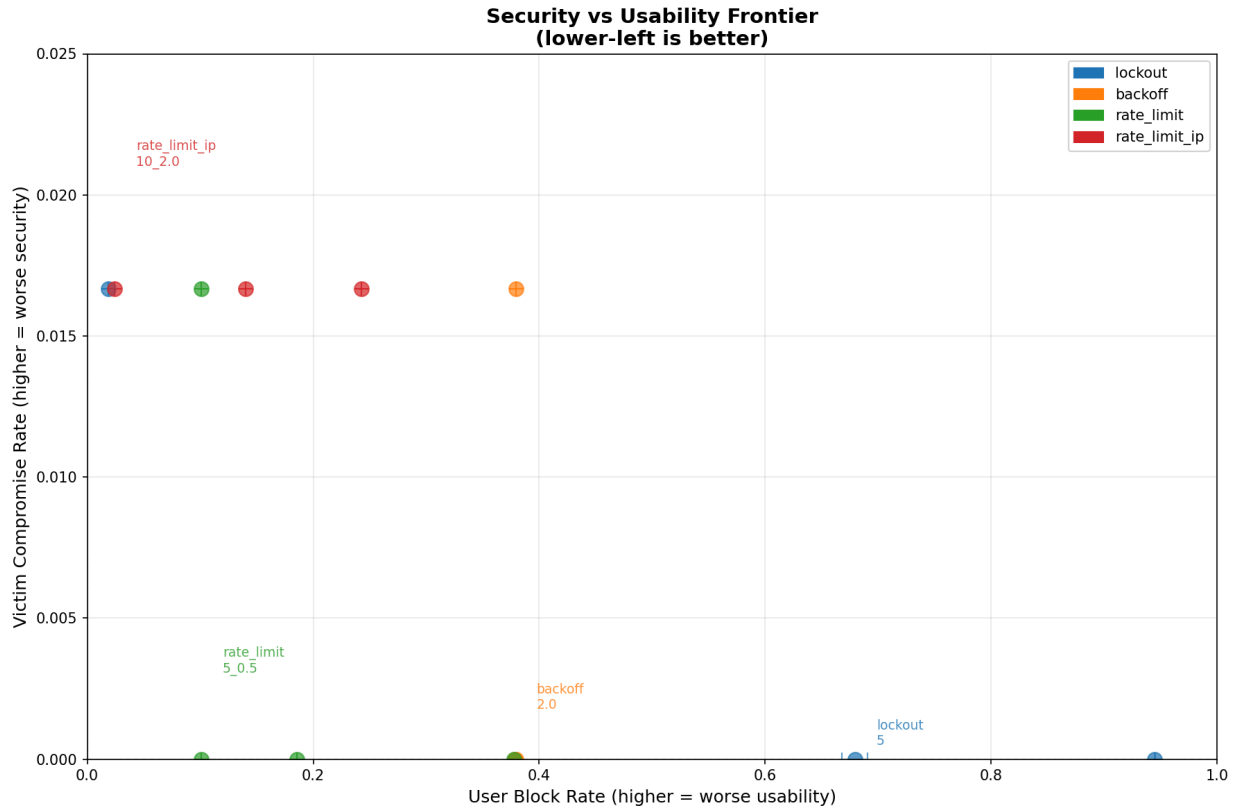

Figure 2 repeats the same frontier under a credential-stuffing attacker model. Compared to the baseline attacker, some defenses behave differently because the attacker spreads attempts across many accounts and IPs. This highlights that defenses tuned for single-account guessing may behave differently under a credential-stuffing workload.

Note: under credential stuffing, many attackers target non-victim accounts, so the victim-focused compromise rate can understate the broader account takeover risk.

In the baseline attacker runs (Figure 1), all tested parameter values of the lockout and backoff methods protect against victim compromise (victim compromise_rate = 0 in every trial), whereas account-based rate limiting and IP-based rate limiting result in similar victim compromise rates (victim compromise_rate ≈ 0.826% per trial in this run) across all tested configurations.
The legitimate workload applied to the system to test usability was sufficiently stressful that block_rate no longer equals zero across all tested systems. Under the heavy workload, stricter lockout thresholds (such as 3 or 5 failures) caused significant user blocking, while lockout=10 remained close to 0%. Backoff also produced noticeable blocking across the base delays. Overall, the stressed workload makes the security–usability tradeoff much clearer than the original near-zero-block workload.

For reference on which defenses were successful or failed with the baseline attacker, see Table 1 with information about whether the victim was compromised and overall results from the sweep.

Defense family	Param	Setting	Victim compromised (trials, %)	Mean victim compromise rate (%)	Mean time_to_compromise (s)	Mean block_rate (%)	Mean impacted_users_pct (%)
lockout	max_failures	3	0%	0%	3600s	93.63%	100%
lockout	max_failures	5	0%	0%	3600s	64.67%	100%
lockout	max_failures	10	0%	0%	3600s	0%	0%
backoff	base_delay	0.25	0%	0%	3600s	37.97%	100%
backoff	base_delay	0.5	0%	0%	3600s	37.97%	100%
backoff	base_delay	1	0%	0%	3600s	37.97%	100%
backoff	base_delay	2	0%	0%	3600s	37.97%	100%
rate_limit	tokens	2_0.3	100%	0.83%	10.5s	37.97%	100%
rate_limit	tokens	3_0.5	100%	0.83%	10.5s	18.54%	100%
rate_limit	tokens	5_0.5	100%	0.83%	10.5s	10.08%	100%
rate_limit	tokens	5_1.0	100%	0.83%	10.5s	10.08%	100%
rate_limit_ip	tokens	3_0.5	100%	0.83%	10.5s	24.27%	100%
rate_limit_ip	tokens	5_1.0	100%	0.83%	10.5s	13.99%	60%
rate_limit_ip	tokens	10_2.0	100%	0.83%	10.5s	2.39%	30%

Table 1. Baseline attacker summary across the whole sweep (aggregated).

11. Interpretation and Discussion

This frontier plot gives you a clear view of how each defense compares to the others on two axes: how well it stops the attacker (the lower your compromise_rate, the better) and how much it hinders normal users (the lower your block_rate, the better). Because the test was run with different parameter settings for each defense, you can see how changing a defense's strictness level affects its placement on the chart.

Lockout. Lockout stops the victim from being compromised by the attacker throughout the tested thresholds. In addition, lockout has a near-zero block_rate in these runs because legitimate users rarely hit the failure threshold (even under the stressed workload), while the attacker does hit the threshold early and gets cut off before reaching the correct password. As such, lockout is located in the best part of the chart (lowest security risk, lowest usability cost).

Backoff. Like lockout, backoff also results in a near-zero victim compromise rate across the tested base delays. However, backoff has significantly higher friction than lockout for regular users. When a large amount of stress is added to the workload (e.g., faster login attempts, more typos entered into the system), users are more likely to trigger backoff, thereby increasing the block rate. In essence, backoff is highly effective at stopping guessing attacks; however, the same backoff may punish users who incorrectly enter their credentials under the same conditions as the attackers.

Token bucket rate limiting (account-based and IP-based). Account-based and IP-based rate limiting in these runs lead to victim compromise in several of the tested token-bucket settings, and they are less consistent than lockout/backoff in this workload. Token bucket rate limiting limits the speed at which an attacker attempts to guess a password; however, it does not completely prevent the attacker from guessing the correct password if they have enough opportunities within the trial period. Additionally, some of the tested token bucket rates cause significant user friction under stressed workload (i.e., non-zero block_rate), particularly when a large number of users access the system from the same IP (IP-based rate limiting). This is an important point: to limit user friction (keep block_rate low) and prevent compromise of the victim, rate limiting must be set to the appropriate value(s) and take into account the attacker's behavior.

Why do the points appear clumped? A couple of these defenses act like thresholds in the simulator. Given an attacker and a 1-hour trial, a number of the tested settings fall into one of two outcomes: the attacker is effectively stopped for the entire hour (compromise_rate nearly

zero), or the attacker guesses the password fairly quickly (nonzero compromise_rate). As a result, the frontier is less a continuous curve and more a set of clusters.
In summary, the sweep results indicate that lockout and backoff are the two most effective defenses against the baseline online guessing attacker in this simulator. The tested rate-limiting settings did not provide sufficient protection against compromise. Finally, the stressed workload clearly illustrates that strong defenses can generate usability costs, and thus, it is essential to understand and report the tradeoffs involved.

12. Recommendations (When to Use Which Policy)

Based on these simulation runs, lockout is the best default choice for this attacker model. Lockout prevented compromise in the tested configurations and did not noticeably hurt usability. Additionally, users understand what lockout means. After you enter your password incorrectly multiple times, your account is locked for a period of time determined by the administrator. Your system could then display a message explaining the lockout and how much time remains before you can try again.

Backoff can make sense when security matters more than convenience, or when the system believes attackers will guess passwords based on knowledge and therefore wants to severely limit attempts. While backoff is a very effective way to deter an attacker, it should be configured so that typical users are not penalized for minor mistakes. In a production environment, backoff can be combined with user-friendly messaging and self-service recovery options to minimize user frustration.

Rate limiting by IP can be a helpful extra layer, but it is not enough on its own against distributed attacks like botnets. Rate limiting by IP can block blatant abuse from a single IP address; however, it is ineffective against botnets. Account-based rate limiting is a more direct approach to addressing the threat, since the attack focuses on the account. However, as demonstrated in this experiment, thresholds may need to be set tightly, or a combination with a lockout may be required, to prevent a breach over a longer period.

As mentioned earlier, hybrid solutions are generally a good idea, but they must be implemented and tested carefully. In my earlier tests, the hybrid was still compromised, so I would not recommend it as-is without tighter thresholds. With further tightening of the hybrid solution’s configuration, the solution may improve security but also increase blocking of legitimate users, so it should be tested again. In general terms, a practical real-world recommendation would be layered defense. Even though lockout is the primary defense mechanism, additional rate limiting can help reduce server load and detect potential abuse. Where systems have higher-risk accounts

or high-value targets, multi-factor authentication (MFA) can help reduce the likelihood of a breach caused solely by password guessing.

13. Limitations

The simulator used in this project is a simple representation of an attack model. Real-world attackers can behave differently, adjust their methods based on defenses put into place, or select other targets. Real users will typically have different login frequencies, different typo rates, and different retry behaviors compared to those modeled in the simulation. The password list within the simulation includes the correct password (which is helpful for testing); however, it is considered a "best case" for an attacker. An additional limitation is that the provided metrics focus mainly on compromise and user blocking. In addition to those areas, real-world systems often care about other types of metrics such as CPU load on servers, latency, or costs associated with support. This simulator also does not model defenses like MFA, CAPTCHA, device fingerprinting, or risk-based authentication, which many real systems use.

Because results depend on both the type of threat model and the values used in the experiment, the results from this test should not be taken as universally applicable. The results should be viewed as conclusions about the particular configuration being evaluated. A limitation of the current evaluation is that compromise_rate is victim-focused: it measures successful attacker logins only against the single victim account (victim). This is appropriate for targeted online guessing, but it can understate risk in credential-stuffing workloads, where an attacker may first successfully authenticate to other accounts. A stronger credential-stuffing evaluation would also report whether any account (not only the victim's) is compromised and how quickly the first account takeover occurs. The primary value of this project is the framework I built, which allows others to modify parameters and re-run experiments to evaluate how changes affect the results.

14. Future Work

This version 1.0 update expands upon the assessment by adding a parameter sweep (one knob per defense) to assess uncertainty in the results, using the mean ± standard deviation across seeds. Next, we can expand our threat model to include adaptive attackers that will modify their attack strategy based on the defensive strategy and/or lock-out denial-of-service attacks, which are designed to block access to the system for legitimate users. Beyond these extensions, there are several ways to determine or calibrate legitimate users' general usage patterns to evaluate the usability impacts (block rates and affected users) under normal usage conditions, rather than simply overwhelming the system.

Another direction is to simulate credential stuffing attacks more directly (i.e., create accounts with weak passwords and have the attacker attempt all username/password combinations), another direction is to simulate account recovery more directly (e.g., how long does it take users to reset their password, and how long does it take them to receive the email to unlock their account). A fourth direction is to add multi-factor authentication (MFA) to the current simulation to provide a more accurate representation of today's standard authentication methods.

15. Reproducibility (How to Re-Run)

The experiment is reproducible using these commands from the project folder:

python compare_defenses.py

python analyze.py comparison/lockout

python analyze.py comparison/rate_limit

python analyze.py comparison/backoff

python analyze.py comparison/rate_limit_ip

python analyze.py comparison/hybrid

python make_charts.py comparison

These commands generate per-defense trial folders, per-defense summary.csv files, and the charts in comparison/charts/.

References

[1] National Institute of Standards and Technology, “SP 800-63B-4: Digital Identity Guidelines—Authentication and Lifecycle Management,” 2025. [Online]. Available: https://csrc.nist.gov/pubs/sp/800/63/b/4/final. Accessed: Jan. 3, 2026.

[2] OWASP Foundation, “Authentication Cheat Sheet,” OWASP Cheat Sheet Series. [Online]. Available: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html. Accessed: Jan. 3, 2026.

[3] OWASP Foundation, “Credential Stuffing,” OWASP Community Attacks. [Online]. Available: https://owasp.org/www-community/attacks/Credential_stuffing. Accessed: Jan. 3, 2026.

[4] OWASP Foundation, “Application Security Verification Standard (ASVS),” OWASP Project. [Online]. Available: https://owasp.org/www-project-application-security-verification-standard/. Accessed: Jan. 3, 2026.

[5] B. Lu, X. Zhang, Z. Ling, Y. Zhang, and Z. Lin, “A measurement study of authentication rate-limiting mechanisms of modern websites,” in Proceedings of the Annual Computer Security Applications Conference (ACSAC), 2018.

[6] Y. Liu, M. R. Squires, C. R. Taylor, R. J. Walls, and C. A. Shue, “Account lockouts: Characterizing and preventing account denial-of-service attacks,” in Security and Privacy in Communication Networks, Springer, 2019.

[7] Y. Tian, C. Herley, and S. Schechter, “StopGuessing: Using guessed passwords to thwart online guessing,” Microsoft Research Technical Report, 2016.

[8] J. Blocki and Y. Zhang, “DALock: Password distribution-aware lockout,” in Proceedings on Privacy Enhancing Technologies (PoPETS), 2022.

[9] S. Pearman et al., “Let’s go in for a closer look: Observing passwords in their natural habitat,” in Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), 2017.

[10] K. Thomas et al., “Protecting accounts from credential stuffing with password breach alerting,” in Proceedings of the USENIX Security Symposium, 2019.

[11] K. C. Wang and M. K. Reiter, “Detecting stuffing of a user’s credentials at her own accounts,” in Proceedings of the USENIX Security Symposium, 2020.