Failed Authentication Counter Strawman

This document proposes a technical architecture for counting failed password authentication events in order to achieve compliance with InCommon Bronze and Silver Identity Assurance Profiles.

Background

Version 1.2 of the InCommon Identity Assurance Profiles establishes certain standards for the use of credentials. Specifically, for Bronze an attack against an authentication secret shall have less than 1 chance in 1024 of success (§4.2.3.2) and for Silver the attack shall have less than 1 chance in 16,384, as well as at least 10-bits of min-entropy (§4.2.3.3). The profiles refer to NIST Special Publication 800-63-1, Appendix A, for a discussion of Authentication Secret complexity and resistance to online guessing. This strawman will consider the case where simple passwords/passphrases are used to achieve compliance with these requirements -- the use of other technologies such as hardware tokens is out of scope.

Calculating Password Entropy

Appendix A includes a lengthy discussion on password complexity and related issues. A mechanism is provided for calculating the password guessing entropy (complexity) of a password, based on its length and other characteristics. For example, an 8 character password requirement with dictionary checks and composition rules has 30 bits of entropy.

min-entropy is defined as "a measure of the difficulty of guessing the easiest single password to guess in the population", and compliance with this requirement can be achieved using dictionary checks or long passwords. As such, further consideration of min-entropy is out of scope.

Applying Entropy To Determine Maximim Unsuccessful Authentication Attempts

In order to determine how to apply these entropy calculations to the IAP requirements, we actually need to look at NIST SP 800-63 v1.0.2, as the calculation language has been removed from 800-63-1. While superceded, v1.0.2 is still referenced by the current Federal Identity, Credentialing, and Access Management Trust Framework Provider Adoption Process (TFPAP) (v1.0.1, dated Sept 2009), with which the InCommon IAPs comply. Referencing Appendix A of SP 800-63 v1.0.2, we see that to calculate the number of possible unsuccessful authentication events permitted, we solve the equation $2^{b} / 2^{n}$ where *b* is the bits of entropy calculated (30 in the previous example) and *n* is the value permitted by the profile in question (10 for Bronze, 14 for Silver).

Continuing with the previous example of an 8 character password, $2^{30} / 2^{10}$ = 1,048,576 failed authentication events would be permitted for Bronze, and $2^{30} / 2^{14}$ = 65,536 would be permitted for Silver.

Options For Limiting Unsuccessful Authentications

The simplest model for limiting authentications is to count the number of failed authentication events and to expire a credential that reaches the limit. Other options include establishing policies that make it mathematically impossible to reach such a limit, usually based on *x* failures within *y* minutes triggering a *z* minute lockout, with passwords expiring every *n* days. Password Entropy Calculators can help determine such policies. Example compliant policies include

Maximum Failures	Failed Attempts Before Lockout	Lockout Duration	Password Expiration
1048576 (Bronze)	15	~10 minutes	Annual
65536 (Silver)	15	~2 hours	Annual

For comparison, NIST SP 800-63-1 (Table 6) dramatically simplifies this policy selection by requiring passwords consistent with 14 bits of entropy for LOA 1 and 30 bits of entropy for LOA 2, and mandating no more than 100 failed attempts per 30 day period for either.

Different institutions will take different approaches based upon local requirements. This document does not argue one solution is better or worse than another.

Arguments For Simple Counting of Failed Authentications

A simple counter is also a simple policy. For sufficiently large permitted numbers of failed authentication events, the likelihood of an individual reaching the maximum threshold under normal circumstances is exceedingly low, minimizing user impact. Where the threshold is approached (let alone reached), it is likely due to reasons that are desirable to have detected, eg an attack in progress or misconfigured email client.

Lockout policies as described above are typically enforced at the credential store. In a heterogeneous environment with multiple credential stores, such as AD, LDAP, and RADIUS, coordinating these policies across the credential stores is difficult. Instead, the policies may need to be "split". In the previous example, perhaps 5 failed attempts trigger a lockout at any one credential store (since one must assume a concurrent attack against all three credential stores). It is easier to centralize a simple counter across multiple credential stores. A centralized tracking facility also simplifies user support.

While compatible with annual (or other periodic) password reset policies, a simple counter does not *require* them.

Arguments Against Simple Counting of Failed Authentications

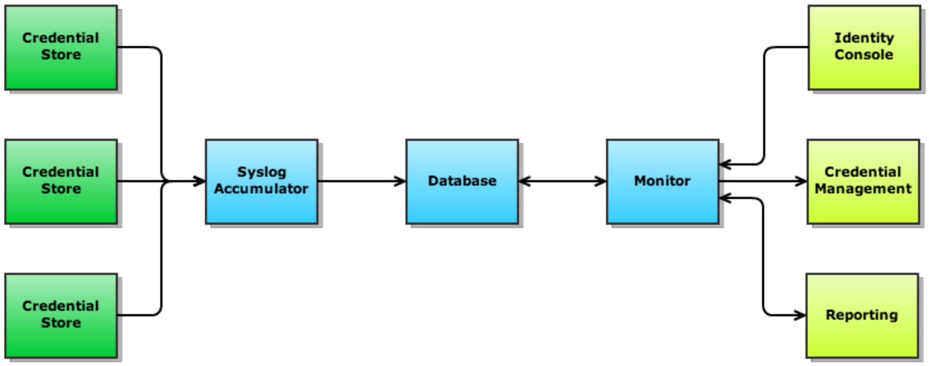
NIST SP 800-63-1 enumerates various problems with maintaining a simple counter (§8.2.3). While some of these are inherent in the highly restrictive policy required (100 failed attempts per 30 day period), some of these can be generalized and are worth some consideration.

- *The approach is subject to denial of service attack.* While true, a lockout is also subject to denial of service attacks, and arguably more quickly.
- *The approach is a single point of failure.* This depends on the actual implementation -- it is possible to invest sufficiently to make each part highly available. Furthermore, if the central counter is offline, IAQs could simply not be asserted for the duration.

- *There may be a delay in the processing of records and the subsequent credential invalidation.* Given that the InCommon IAPs require credential revocation within 72 hours of notification of compromise (§4.2.4.2), it seems reasonable that a delay of up to several minutes (if not longer) is acceptable in processing an invalidation once a threshold has been reached.

The mitigations described in 800-63-1 (requiring CAPTCHA after a few failed authentications, increasing waits between subsequent authentication attempts, whitelisting IP addresses for authentication) are worthy of consideration, but are independent of the approach selected.

Failed Authentication Counter Architecture (Strawman)



This proposal brings together several components to support multiple integration patterns.

Use of Syslog

This proposal relies on the use of syslog as a mechanism to export failed authentication events from credential stores in (near) real time, as it is widely supported natively or can be easily supported via add-ons in many flavors of Kerberos, LDAP, and RADIUS, as well as (pending details) Active Directory.

ℹ Per-credential store instructions are needed for exporting failed authentication records via syslog.
ℹ Some credential store-specific plugins may need to be developed.

Syslog Accumulator

Syslog is a (near) real time protocol, potentially generating a significant amount of traffic. An existing product such as rsyslog (GPL v3) could be used as the accumulator, as it is high performance, can be made highly available, can write to a database, and is modular.

ℹ Optimized modules for failed authentication counting could be developed.
ℹ Reference instructions for installation and configuration would need to be written.

Database

A relatively simple schema can track failed authentication events as reported to the Accumulator. A basic table would track

Column	Description
`rowid`	ID of this row
`subject`	ID of the credential
`service`	ID of the service generating the failure event
`ip_address`	Originating IP address of the failure event
`timestamp`	Time of the failure event

An additional table is required to track when a count should be reset (perhaps due to a password change). A simple method would be to have a separate table to track the current count. Such a table should probably be managed by the Accumulator, as relying on stored procedures can have concurrency implications.

Column	Description
`rowid`	ID of this row
`subject`	ID of the credential
`current_count`	Current count

A table that tracks reset events (relying on database or application logic to calculate the current count) would also be viable. This table would be managed by the Monitor.

Column	Description
`rowid`	ID of this row
`subject`	ID of the credential
`reset_timestamp`	Timestamp of count reset event

Monitor

ℹ The Monitor would likely need to be developed (presumably as an Open Source project).

The final component is the Monitor, responsible for determining when thresholds are reached and taking appropriate action, as well as for managing reset events. The most portable approach may be for the Monitor to periodically poll the database for counts, and then fire off events as appropriate. (It may be possible to leverage database trigger mechanisms to push events to the Monitor, but this is unlikely to be achievable in a cross-platform way.)

See Also: Monitor Requirements