Secure Password Storage

This document describes goals, threats, and design for a reusable password storage module (PSM or module). It describes module goals in terms of 1) compliance to best practice, policy, and known applicable regulatory demands, 2) configuration for use, 3) integration and use with JEE systems (including framework support and dependencies), 4) security properties intended to hold during runtime use, as well as 5) design and behavior for maintenance within a deployed system.

This document can be found at: http://goo.gl/Spvzs

Author: John Steven (john.steven@owasp.org)
Contributors: Jeff Walton, Kevin Wall
Reviewers: Chandu Ketkar, Scott Matsumoto

Version	Date	Author	Comments
v0.1	2012-06-12	jsteven	Initial Draft
v0.2	2012-06-21	jsteven	Integrated comments from Kevin Wall, Amit Sethi, & Jim Delgrosso; Finalized draft "Problem Definition"
v0.3	2012-10-05	jsteven	Integrated commentary from Chandu Ketkar, Scott Matsumoto
v0.4	2012-11-01	jsteven	Integrated (implicit) feedback from presentation @AppSecUSA 2012

Table of Contents

Compliance

At a high level the password storage module (PSM) must both comply with best practices for resisting attack and also with stated best practices. Presently OWASP appears to have no stated "programming standard" to which PSM can adhere. Best practices from across the OWASP organization apply. Some include:

- Password Storage Cheat Sheet, Cryptographic Storage Cheat Sheet
- PKCS #5: RSA Password-Based Cryptography Standard
- Guide to Cryptography
- Kevin Wall's Signs of broken auth (& related posts)
- John Steven's Securing password digests
- IETF RFC2898, RFC5869
- scrypt, IETF Memo: scrypt for Password-based Key Derivation

Less specifically, the following apply:

- Secure Coding Principles
- OWASP SCP Quick Reference Guide

Other work

- Spring Security, Resin
- jascrypt
- Apache: HTDigest, HTTP Digest Specification, Shiro

Applicable Regulation, Audit, or Special Guidance

- COBIT DS 5.18 - Cryptographic key management
- Export Administration Regulations ("EAR") 15 C.F.R.
- http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf
- Future work:
    - Recommendations for key derivation NIST SP-800-132
    - Authenticated encryption of sensitive material: NIST SP-800-38F (Draft)

ESAPI

PSM explicitly does not attempt to upgrade or integrate with ESAPI 1.x or 2.x. PSM's authors do intend for resulting design implementation to be appropriate as a candidate for the ESAPI modularization initiative.

Compliance Goals

PSM will reflect and adhere to guidance provided by the above sources unless otherwise explicitly refuted or otherwise addressed.

High-level Goals

The PSM intends to support the following properties:

- [EBS]-Entity-based storage - PSM will provide memory-based storage of passwords to be used for AuthN. PSM implements a default SQL-based database to support stand-alone installation and usage.
- [FI]-Framework Integration - PSM intends to integrate with popular JavaEE frameworks, including Struts 2.x & Spring 2.x & 3.0. The preferred integration model is configuration for use and extension where necessary. If necessary, PSM will use dependency injection or patching to 'retrofit' a framework.
- [BI]-Build Integration - PSM will provide a Maven-based build chain for easy construction and maintenance. Maven targets will exist for operations such as construction of cryptographic material (such as keys, seeds, and so forth).
- [SD]-Secure By Default Operation - When PSM provides configuration, its operation should be, by default, the most secure offered.
- [GA]-Grow with Adopter - though PSM supports easy creation of "hello world" example programs PSM intends to facilitate use by complex applications and will not preclude applications from leveraging Enterprise-class commercial Identity and Access Management (IAM) products. PSM intends to provide scalable, low-latency operation. Specifically, PSM should support simultaneous access by hundreds of thousands of users amongst a population of millions of users with a latency no greater than one (1) second.
- [AR]-Attack resistant implementation - PSM intends to resist those attack vectors described by the threat model.
- [OR]-Support for On-line Key Rotation - PSM must provide facilities for rotation of algorithms, cryptographic material, and subsequent migration of password stores during normal operation. Rotation/migration may occur as a result of policy (key rotation), incident, or due to degradation of an approach (key strength, advances in cryptanalysis, etc.). Rotation may also occur in promotion to particular deployment environments (dev, test, staging, prod).
- [BP]-Adherence to programming 'best practices' - the implementation is to adhere to best practices for both security and quality including conventions, documentation, unit testing.

PSM is not a fully implemented stand-alone authentication or IAM system. As such, PSM will not provide user/credential management, password fitness or reset, or other features essential to the security posture of a system. Where such features require direct interaction PSM will provide interfaces or stubbed functionality.

PSM does not solve problems intrinsic to digested passwords (e.g. replay of observed tokens) or problems related to passwords (such as reset or guessability). We offer PSM only as a better design/implementation of an outmoded approach.

More recent approaches resist attack better than PSM. Specifically, this work acknowledges OAuth, OpenID, and SAML; but these recommendations do not conform to the cited standards. Those undertaking this work recognize the superiority of alternative approaches in creating, exchanging, and validating claims about identity and security.

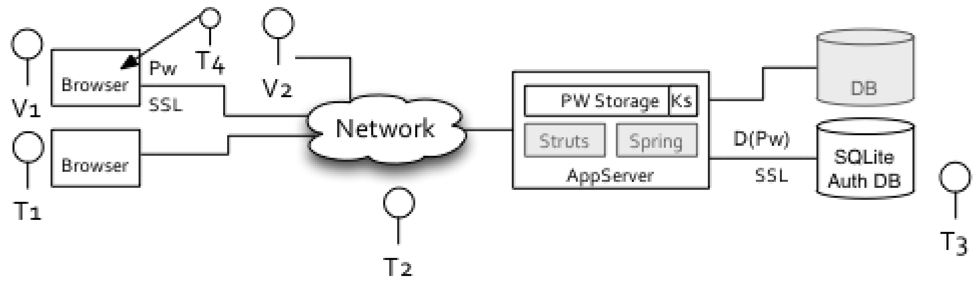
Threat Model

In many cases, the threat model necessitates a detailed *attack surface* diagram. However, in this case, modeling what data an attacker has access to suffices. For the purposes of PSM, we break the threat model into two parts:

1. ***Acquiring*** a [protected] password database and material necessary to reverse it;
2. ***Reversing*** the [protected] database, once stolen, to reveal credentials.

The same population of threats takes part in both of the above attacks. However, a certain threat acquiring the database may rely on another (perhaps more capable threat) to reverse the protected material.

In the figure below threats and a victim decorate a common deployment model used as a the basis threats, their capabilities, and access to attack surfaces. (for more detail, see the “Deployment Models” section).



The above deployment separates application server from database. For the purpose of this model separation entails deployment on separate hosts and no shared login/ credentials/permissions between each administrator (appserver & database admin).

Notes on single-server deployment

The distinction between application server and database, as well as their respective administrators holds in every large deployment with which this document’s author currently works. However, systems fielded by smaller organizations may choose to either deploy application server and database on the same host, or grant a single administrator access to both. Where possible this document will explicitly call out mitigations for the challenges facing a single-server environment.

Some proposed solution configuration rely on design principles “Compartmentalization” and “Separation of Privilege”. This remains possible in a single-server deployment by relying on separation between the production environment and some other trust zone. Configuration management, deployment engineering, or (depending on configuration and deployment) an HSM can define such a separate zone protected from threats.

No silver bullet for secret storage/management exists. If an organization can not define any “completely trusted” zone and is not large enough to separate two partial trust zones then it will not be possible to completely protect against insider threat.

Threats

The following actors participate in this threat model:

1. [V1] - *Active System User* : Compromises one’s self through use of the system
    a. Accesses the system normally through a browser
    b. May access the system through a compromised network (exposing them to [T2])

2. [V2] - Offline *System User* : Suffers compromise without use of system
    a. Accesses the system normally through a browser but may remain inactive
    b. May suffer compromise due to by duplicate password detection or by bulk export and reversal of stored passwords, for instance.

3. [T1] – *Internet-based Threat* : Access to the app
    a. Capabilities - ***acquiring & reversing*** passwords
        - Possesses [1+] valid login to system;
    b. Capabilities - ***acquiring*** passwords
        - Interacts with system through browser or AppSec tools;
        - Capable of discovering/executing AppSec attacks;
        - Capable of acting as MiM [T2] in addition to T1 capabilities;
        - Data show that if [T1] can lift PW, they are likely to also be able to lift/correlate UNs;
        - Can NOT conduct effective bulk phishing, malware installation, or botnet campaigns beyond individual AppSec attacks (i.e. CSRF, etc.);
    c. Capabilities - ***reversing*** passwords
        - Has access to personal or commercial-grade hardware. This may include:
            - Machines with multiple graphics cards
            - Limited CPU/GPU time on a cloud provider (such as EC2)
            - Configurations <= $10,000
            - Access to <= 20TB of storage
            - Access to <= 512 compute units
            - Time limited by attention

        - Capable of:
            - Dictionary attack
            - Brute-force attack
            - (pre-computed) Rainbow Table attack
        - Can “hand-roll” or write tool-based brute-force attacks;
        - Has access to downloadable hardware-accelerated brute force attacks (i.e. GPU cracking tools) that can crack SHA1 and similar hashes at XXXGB/s bandwidth;
        - Can not crack adaptive hashes (PBKDF, bcrypt, etc.) at similar throughput;
        - Can NOT conduct state-of-art (accelerated/optimized) cryptanalytic, statistical, or create rainbow-table attacks;
        - Should NOT be considered capable of Man-in-Browser (MiB) attacks.

    d. Goals
        - Fame - Create hype/name through scheme reversal, partial DB breaks, & similar
        - Single break - Given a population of users, reverse one (any) user’s password
        - Particular break - Impersonate a particular user [V1]. Target may be of opportunity (such as in a cafe) or choice (celebrity).
        - Monetary gain - Reverse a set of users’ credentials [V2] with hopes of sale. Success, in this case, would likely not be advertised.
        - Reputation damage to site owner
        - Use of system A credentials in system B for a particular user [V1].

    e. Attack Vectors
    Acquiring:
        - [AVA00] - Observes client-server interaction over the network;
        - [AVA01] – Inject DB, lift entire table of <protected>(PW);
        - [AVA02] - Successfully expose Victim [V1] to scripted AppSec attack, resulting in XSS, CSRF, or SQLi. Such an attack may occur prior to or post [V1] authentication;
        - *** If [T1: Internet-based Attacker] successfully executes [AV3] against all [V1]’s (or [V2]) ‘promote’ [T1] to [T4: MiM].
    Reversing:
        - [AVR01] – Use any authentication (authN) API exposed by the server directly to attempt brute forcing PW
        -

4. [T2] - *Man-in-the-Middle* : Ability to interpose in communications between victim [V1] and the application server serving it content.
    a. Capabilities - ***acquiring*** passwords
        - May passively observe HTTP traffic;
        - May actively observe and modify HTTP traffic, as a proxy;
        - May passively observe SSL traffic (HTTPS);
        - May be able to interpose, observe, and modify SSL as a proxy;

- Capable of network-based attacks but not able to ‘break’ SSL in new or innovative ways;
- Capable of replaying observed traffic;
- *** If [T2:MiM] conducts [T1.AVA02] modifying code/script bound for [V1], they ‘promote’ to [T4:MiB]. See [T4].

b. Capabilities - ***reversing*** passwords
- See [T1].

c. Goals
- Fame - See [T1]
- Particular Break - See [T1]
- Impersonation - See [T1]
- Monetary Gain - See [T1]

d. Attack Vectors
Acquiring:
- [AVA03] - Interposition / Proxy attacks: observe traffic, parse, and understand protocol, elements of request/response, and state;
- [AVA04] - Interpose in SSL traffic (by owning a network segment) for some set of victims [V1];

Reversing:
- [AVR02] - Timing Attacks (where applicable): Attacker uses relative difficulty of successfully verifying stored PW to determine whether a known input (PW or digest) is correct.

5. [T3] – *LAN-based Threat* : Threat actors within equivalence-class to DB admin

a. Capabilities - ***acquiring*** passwords
- May acts as [T2] within the network segment AppServer ← → DB;
- Has console/network access to database;
- May have access via access control lists (ACLs) to AuthN credentials unless otherwise specified;
- Presumed to be ‘root’ on database;
- May index, sort, and conduct other operations on bulk <protected>(pw) store ‘invisibly’.

b. Capabilities - ***reversing*** passwords
- See [T1].

c. Goals
- See [T1].

d. Attack Vectors:
Acquiring:
- [AVA05] – Bulk export of material through load/reporting interfaces;
- [AVA06] - Use of [T1] access to application (through registration) post [T3.AV01];
- [AVA07] - Unauthorized theft or restore of credentials store from backup;

        - [AVA08] - Access to credentials logged on server or aggregator or off-site backup provider.

    Reversing:

        - [AVR03] - Can, unless otherwise protected (through salts, similar) sort and detect duplicates in <protected>(PW) database;

6. [T4] - *MiB Threat* : [T1] with access to victims' browsers
    a. Capabilities - ***acquiring*** passwords
        - See [T1];
        - Capable of conducting [T1.AVA02] (replace code-in-browser);
        - Capable of adding persistent code/data to victims' [V1] browser.

    b. Goals
        - See [T1].

    c. Attack Vectors

    Acquiring:

        - [AVA09] - Lands keylogger prior to [V1] accessing site;
        - [AVA10] - Lands persistent script prior to [V1] accessing site

7. [T5] - *Concerted Threat* : [T1]-like threat with capabilities of [T1], [T3], and [T4]. LAN access obtained through means of compromise of other or related systems.
    a. Capabilities - ***acquiring & reversing*** passwords
        - Well-funded, patient threat has unlimited time/money
    b. Capabilities - ***reversing*** passwords
        - Capable of cryptanalytic attack, including but not limited to:
            - Dictionary attack
            - Brute-force attack
            - Rainbow Table attack
            - Length-extension attack
            - Padding Oracle attack
            - Chosen plaintext attack
            - Crypt-analytic attack
            - Side-channel, timing attacks
            - ***Note: depending on scheme, some of above are N/A
        - Access to handwritten hardware accelerated attacks at XXXGB/s throughput, including adaptive hashes (through use of FPGA or other custom hardware).

    c. Goals
        - See [T1];
        - Fatal reputation damage to site owner;
        - Terrorism, cyber-war.

    d. Attack Vectors

    Acquiring:

    - [AVA11] - Infrastructure attacks (DNS, Certificate Authority compromise, etc.);
    - Compels [T3] to acquiring database through his/her attack vectors;

  Reversing:
    - [AVR11] - Registers two users [$T1_A$] and [$T1_B$], discerns properties of AuthN system. In concert with [T1.AVA01], executes “chosen plaintext” attacks with multiple chosen plaintexts;
    - [AVR12] - Using [T5.AVR11] (or similar) uses timing attack (where applicable) on AuthN system to determine properties of successful or failed login, such as round-trip-time;
    - [AVR13] - Constructs rainbow table in order to reduce time/space requirements compared with brute-force style attack;
    - [AVR14] - Conducts oracle-padding, length extension, or similar attack on cryptographic primitive;
    - [AVR15] – PRNG attacks: demonstrate improper use of PRNG leading to cryptanalysis or exploit;

  DoS:
    - [AV*01] - DoS attack: exhaust randomness preventing server response.

Note: A formal threat model would present attack vectors as attack trees anchored by each threat at attack surfaces. This document lists attack vectors numerically for ease of reference. The reader, by no means, should consider this list exhaustive.

Rules of Engagement

Based on the threat model described above the following rules-of-engagement govern the application of attack vectors, determination of successful exploit, and validity of mitigation strategies.

- [T1], [T4], or [T5] may seek a partial PSM break in order to discredit site owners even without a full compromise or bulk credential compromise. We consider this event “complete success” for the attacker;
- Grading difficulty of breaking any single user’s credentials considered in terms of average difficulty. That is, on average, a threat can break at least one user’s password on the order user_population / 2;
- Grading a break on a particular user’s credentials considered in terms of worst-case difficulty (rather than average) at O(user_population). Threat subject to additional difficulty as username not stored with passwords;
- [T1], [T4], or [T5] may discover a break that succeeds with some probability < 1. In this case, if the probability of success >= 10% over a population of <= 2,000,000 users exist, we consider this event “complete success” for the attacker. That is, if the threat can reverse 200,000 credentials, they are considered successful;
- PSM ‘resilience’ graded in terms of site owner (defender) ability to continue providing services in the face of [T1.AVA01] or [T3.AVA05] (bulk compromise of

<protected>(PW) for all [V2]) through goals of key/scheme rotation [SOKR] & compensating controls. If the defender must deny users (>= 10% of population) access to the compromised site we consider the attacker "successful" in DoS;

- PSM must provide protection for user credentials in the face of [T1.AVA01] or [T3.AVA05], even against [T5] - *concerted threats*;
- PSM must provide comparable protection for a victim [V1]'s PW even if the PW suffers due to poor fitness;
- Any attack resistance provided by PSM should be provided in stand-alone operation and in operation integrated with all supported frameworks;

The following table maps threats and their attack vectors to rules of engagement:

Threat	Attack Vector	In-bounds?	Comments
[T1] AppSec			
Acquire	AVA00 - Observe client operations	Yes	
	AVA01 - Inject DB, bulk credentials lift	Yes	
	AVA02 - AppSec attack (XSS, CSRF, SQLI)	Yes	Stub AuthN integration
Reverse	AVR01 - Brute force PW w/ AuthN API	Yes	
[T2] MiM			
Acquire	AVA03 - Interposition, Proxy	No	Plaintxt creds. Game over.
	AVA04 - Interposition, Proxy, SSL	No	See above.
Reverse	AVR02 - Timing attacks	Yes	
[T3] Admin			
Acquire	AVA05 - Bulk credential export	Yes	
	AVA06 - [T1] style attack	Yes	
	AVA07 - Direct action w/ DB	Yes	
	AVA08 - Unauthorized restore from backup	Yes	
	AVA09 - Access logs, aggregator, off-site	Yes	
Reverse	ARV03 - Sort or detect duplicates	Yes	
[T4] MiB			
Acquire	AVA10 - Keylogger	No	Plaintxt creds. Game over.
	AVA11 - Other persistent script/data	No	See above.
[T5] Concerted			
Acquire	AVA12 - Infrastructure attack	No	Credentials seen plaintext
Reverse	AVR04 - Register 2 users, compare	Yes	

	AVR05 - Timing attacks via [T5.14]	Yes	N/A for solutions herein
	AVR06 - Rainbow table, space/time saving	Yes	
	AVR07 - Padding, length extension attacks	Yes	N/A for solutions herein
	AVR08 - PRNG Attacks	Yes	
	AV*01 - DoS (e.x. exhaust randomness)	Yes	

If unclear or unstated, rules of engagement adhere to the following rule: PSM makes every attempt to protect (PW) data at rest and only protects (PW) data in transit to the extent that <protection>(PW) has already been applied.

Out of Scope

The table above indicates situations considered out of scope by PSM using greyed out cells. Specifically, PSM provides ***NO*** protection against compromise where:

- Credentials passed from client ← → server in plaintext form as observed by [T2];
  To meet this goal consider:
  - Properly encrypted client-server communications;
  - Challenge-response schemes (Oauth, SRP);
  - One-time pads
- Credentials entered in [V1]'s browser can be observed by [T1] or [T4];
- Threats [T5] successfully phish credentials from victims [V1].

This approach originally considered keeping attack resistance in the face of SSL failure between client ← → server *in scope* but that was deemed inappropriate for a password *storage* module by itself.

Scope: Detailed Goals

This section describes goals for PSM design and implementation. In a more formal software development effort, statements made in this section constitute requirements.

- STR-1.1 : Support a memory-based entity model designed to represent and store user credentials in protected form.
- STR-1.2 : Build scaffolding for creation of a SQL-based database to support stand-alone usage
  - 1.2.1 : Setup of initial credential store;
  - 1.2.2 : Tear-down of offline database credential store (cleanup).
- STR-1.3 : Support storage of versioning information for the PSM scheme
  - 1.3.1 : Version stamping on password creation
  - 1.3.2 : Validation of scheme version on password verification
  - 1.3.3 : Verification of former versions
  - MTC-1.1 : Support for online stored format rotation including:
  - 1.1.1 : Scheme (i.e. cryptographic protocol)

  - 1.1.2 : Algorithm (e.g. adaptive hash, hmac, etc.)
  - 1.1.3 : Cryptographic parameters (e.g. K_{s1})
  - 1.1.4 : Migration (Online/Offline) of database on format change;
- MTC-1.2 : Design for compartmentalization and separation of privilege for secret material (such as MAC or encryption keys) between AppServer and Database;
- ALGO-1.1 : Support:
  - 1.1.1 : Pluggable password protection schemes
  - 1.1.2 : Configuration-based protection scheme specification
- OOD-1.1 : [default] Support for "hardened" mode, including:
  - 1.1.1 : Resist server (JEE) heap-dump attacks
  - 1.1.2 : Resist side-channel attacks, where applicable
- TST-1.1 : Support for a testing interface with predictable values in place of random values to facilitate test oracle operation;
- CAM-1.1 : Imposition of a disproportionately greater impact on attacker compared with the defender, their users, and victims.
- KCS-1.1 : Security posture can independent of obscurity. That is, knowing the protection scheme's details should not result in scheme compromise.
- ATK-1.1 : Prevent bulk exfiltrate of PW DB data to web through application (*See [T1.AVA03], [T3.AV07]*);

Specific Basic Cryptanalytic Concerns

Regardless of what cryptographic primitives solutions employ, basic attacks commonly apply. PSM design and implementation seeks to resist failure due to these attacks explicitly by possessing the following properties:

- ATK-1.1 : Resist "chosen plaintext" attacks - Attackers possessing system access and a valid account [T1] (*See [T1.AVA01], [T5.AVA11]*) should ***not*** be able to:
  - Discern password protection scheme
  - Attack another user [V2] in O($V1_{pw}$) time
  - Choose and use set(s) of credentials and discern scheme cryptographic secrets
- ATK-1.2 : Resist "brute-force" attacks - Attackers possessing access to PW DB and knowledge of protection scheme (*See [T1.AVA02]*) should not be able to:
  - Discern individual account credentials in reasonable time
    - ■ Difficulty >> O($V1_{pw}$)
    - ■ Calendar time >= 1 yr
  - Discern all account credentials in reasonable time
    - ■ Difficulty >> O(V_{pw}) * Population(V)
    - ■ Calendar time >= 1 yrs
- ATK-1.3 : Resist D.o.S. as a result of entropy/randomness exhaustion (*See [T5.AVR08]*);
- ATK-1.5 : Resist identifying identical credentials by observing <protected>(PW) (*See [T1.AVA00], [T3.AVR03, T5.AVA12, T5.AVR04]*);
- ATK-1.6 : Prevent attackers from generating valid forms <protected>(PW) without knowing credentials and possessing any/all secrets;

- ATK-1.7 : Prevent attackers from exfiltrating any ancillary secrets associated with <protected>(PW), such as MAC or encryption keys (*See [T3.AVA05-T3.AVA09]*);
- ATK-1.8 : Prevent attacks from gaining information about plain/digest-text through side-channel or timing attack: for instance, gauging how long equality check between two digests takes (*See [T5.AVR05]*); [*TA][1]
- ATK-1.9 : Prevent attackers from crafting a extended plain-text that collides with [V1] digest w/o knowing V1 plaintext password (i.e. length extension attacks or those attacks seeking to influence a final block fed to mac/cipher function) (*See [T5.AVR07]*); [*LE][2]

Specific Subtle Cryptanalytic Concerns

The use of certain cryptographic primitives carries properties attackers may use to unexpectedly defeat the system, either directly or by making other attacks (such as brute force) more tractable. PSM design and implementation seeks to resist such failure explicitly through possessing the following properties:

- SCC-1.1 : Prevent attackers from gleaning information about server secrets or [V1] plaintext through multiple chosen plaintexts (such as (PW, PW') and (PW', PW'') : PW' = digest(PW)); [RG][3]
- SCC-1.2 : Prevent attackers from gleaning information due to use of a common key between cipher and mac constructs, such as when CBC-MAC used; [HA][4]
- SCC-1.3 : Prevent leakage of information (such as password, key material, initialization vectors, etc.) when using cryptographic ciphers, hashes, or MACs.
- SCC-1.4 : Assert that input to cryptographic primitives possesses the appropriate level of randomness without imposing such undue requirements on the system so as to easily exhaust its entropy thus denying service;
- SCC-1.5 : Bound input to those primitives which fall prey to length-extension attacks;
- SCC-1.6 : Take care to avoid padding oracle attacks where applicable;
- SCC-1.7 : Take specific steps to prevent primitives from leaking information about plaintext or keys when attackers have access to plaintext/ciphertext pairs.

High-level Design

Initial PSM design relies on a few key components designed to meet the previous sections' detailed goals, rules-of-engagement, and threat model. Though PSM intends to provide only storage its design demands both structural and behavioral elements

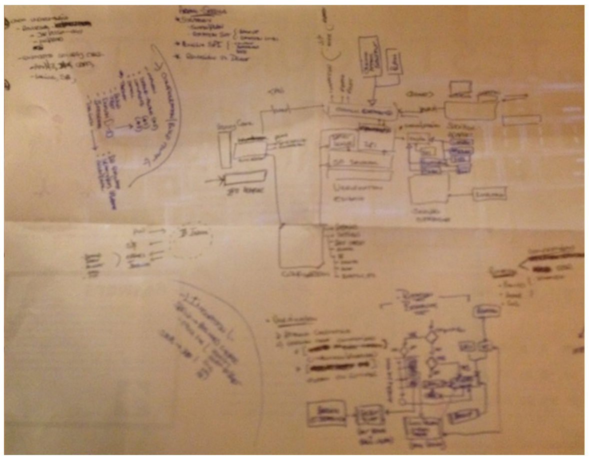
[1] [TA] - *OpenJDK MessageDigest.isEqual Introduces Timing Attack Vulnerabilities*, https://bugzilla.redhat.com/show_bug.cgi?id=530057

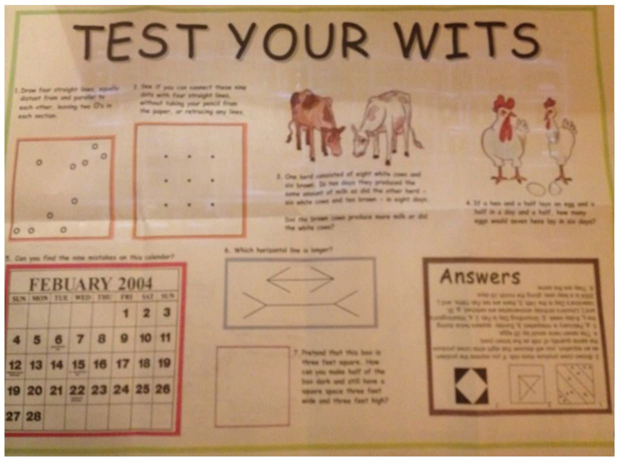
[2] [LE] - *The Security of Cipher Block Chaining Message Authentication Code*, http://cseweb.ucsd.edu/~mihir/papers/cbc.pdf

[3] [RG] - http://www.cs.berkeley.edu/~daw/papers/ddj-netscape.html

[4] [HA] - Handbook of Applied Cryptography, Schneier, p367

to handle 1) scheme and algorithm selection and 2) scheme/key rotation on appropriate triggers (including active attack). Figures below show the initial design:





PSM Design - "One-pager" (FRONT) PSM Design - "One-pager' (BACK)

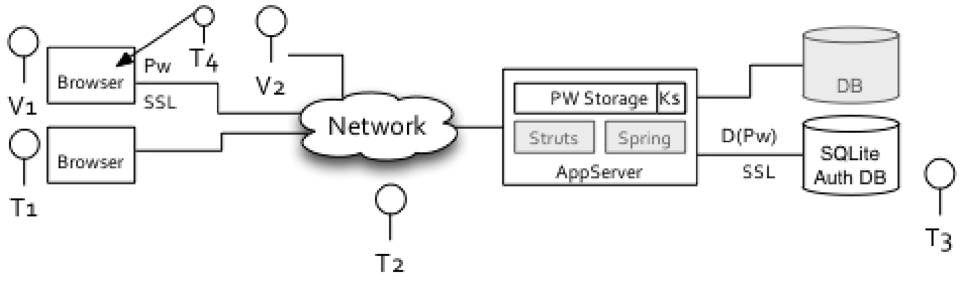
Key components of the PSM design pictured above include ("||" denotes concatenation):

- Strategies, algorithms, & template-patterns
  - Credentials verification
  - Engine - <protect>PW :: Reversible | Adaptive-only | HMAC-Only
  - Rotation :: EoL | Compromise | Policy
    - Backup
    - Upgrade Worker-thread
  - AuthN Barrier :: Verify() | Alt-Verify()
- Façades, APIs, adapters, & proxies
  - Configuration
  - AuthN: Credentials input, result
  - Cryptographic SPI :: COMPAT/FIPS | MODERN
  - Intrusion Prevention/Detection
- Structural Patterns & Entities
  - <protected>PW :: Versioned | V'd-Digest | Ciphertext | V'd-Ciphertext
  - Keystore
    - Scheme entry

- Cryptographic material

Deployment Model & Dependencies

Standard server-side deployment involves a library (.NET assembly, Java jar) containing functionality and entity bridge to a separate stand-alone authentication database. PSM key storage occurs within the application server.



Any scheme in which PSM extends protection against PW compromise to the browser-to-server connection in insecure areas (or those experiencing SSL compromise) is considered out of scope (See “out of scope” section for more information).

In order to meet needs of a wide range of adoption scenarios (from green-field “hello world” to integration within existing enterprise applications) PSM supports the following deployment models:

Deployment		Configuration	Prescription
Stand Alone			No existing application DB; “Hello world”
Integration			Adopter uses PSM in existing application
		Supported by both Stand-alone and integration	
		Rotation ENABLED or DISABLED	When enabled PSM rotates <protected>PW due to: EoL of algorithm / key-strength, policy, or as attack response
		REVERSIBLE, ADAPTIVE, HMAC	Engined used to derive <protected>PW. This determines 1) behavior under attack, 2) compromise due to comprehensive insider threat, and 3) whether/ when user interaction necessary for pw/scheme update
		COMPAT/FIPS, MODERN	PSM providers honor two classes of protection: COMPAT/FIPS (older platforms such as Java5) and modern (such as Java6+)
		Crypto Provider (SPI)	Specific JCA implementation: IBM, Sun, AAPL, …

Adopters can “quick start” by choosing “stand-alone” deployment, disabling *rotation*, and using the HMAC engine with a *supported* crypto primitive provider.

*nix Environment Support

PSM deployment support planned on Linux 2.6.4+ and Mac OS X 10.8.x or greater.

- Platform
  - Tested:
    - >= Ubuntu 12.04 “Precise Pangolin” or greater
    - >= Apple OS X 10.8.1 “Mountain Lion” or greater
- Java JDK
  - Oracle, OpenJDK, or Apple: Java 6,
    - “SUN”, “SunJCE”, “SunJSSE” JCA crypto-provider(s)
      - SHA1PRNG, HmacSHA256, PBKDF2WithHmacSHA1
  - IBM Java 6
    - IBMJSSE2
      - SHA1PRNG, HmacSHA256, PBKDF2WithHmacSHA1
    - OPTIONAL: IBMJCEFIPS
      - SHA256, HmacSHA1, FIPS186-2 (Randomness)
- Apache Maven 3.0
- JUnit 4.x
- SQL Database (comes with SQLite by default)
- UTF-8 Strings

Optional dependencies for framework integration include:

- MVC Framework
  - Struts 2.x
  - Spring 2.x, 3.x
- MySQL Database
- Javascript-enabled Browser

Compatibility Statements

Though PSM requires Java6 or greater, PSM will provide reduced support for Java JRE5/ JDK5 (Sun, IBM) in a COMPAT/FIPS ‘compatibility mode’. Implementations of Java5 do not possess modern cryptographic primitives essential to PSM. Rather than ‘rolling its own crypto’, PSM design makes due with those FIPS-certified primitives available to the ‘compat’ SPIs. PSM takes reasonable steps to increase attack resistance in COMPAT/FIPS mode but the scheme can not be relied upon to provide similar attack resistance to the MODERN implementations.

PSM provides no support for Java 1.4.x and will not compile on Java 1.4.x or earlier.

Individuals have promised to donate Windows-based JEE and .NET ports. A Unix-based C++ port may be defined. However, it is currently unclear what framework integration that entails.

Key Tooling/Storage

Certain solutions (reversible and COMPAT/FIPS) rely on a cryptographic key to provide some or all of their attack resistance. In each case, these solution designs will refer to a “PSMKeyTool”. The PSMKeyTool represents an abstraction designed to handle the non-trivial tasks of key storage, use, and management between environments (dev, staging, prod). PSM *must* provide PSMKeyTool implementation in order to prevent likely developer error, thus defeating the scheme’s security proposition.

Detailed Design

Alternative Approaches

The number of differing approaches to password protection demand that any design explicitly document its tradeoffs relative to alternatives. Evaluation of alternative approaches hinges on requirements from the detailed design goals section and the list of attacks laid out by the threat model and rules of engagement.

Adaptive Hashes
Security practitioners commonly select adaptive hashes as their preferred solution to password storage. Two contenders (PBKDF2, bcrypt) dominate popular suggestion with a third (scrypt) often left as an exercise to the reader. This document addresses all three explicitly.

Protection Engine Algorithms

PSM specifies three engine configurations in order to support the following constraints, as any of each may apply in a given environment. The solution must be:

- COMPAT/FIPS - Legacy runtime compatible and FIPS compliant;
- Reversible - Providing strong protection, compartmentalization, and separation of privilege;
- Adaptive Only - Supported through Enterprise approved JRE, providing strong protection at some expense to defender in terms of scale and maintainability.

This document specifies each solution as an algorithm and then provides a series of explanation in subsequent sections. First, a brief explanation follows each addressing high-level decisions. Second, a section offers considerations made against both basic and subtle cryptanalytic attacks outlined previously by this document. The following assumptions hold for all attack vectors:

- Dictionary-based attacks represent O(2_{21}) -or- (2,000,000) entries
- "Exhaustive" searches of "low fitness" PW routines represent O(2_{186}) tries, or the search space of a-zA-Z0-9 x over a search space of eight (8) characters.
- GPU-based attack represents a speed of O(2_{32}) -or- (4,500,000,000) tries per second. GPU-based attack demands exhaustive search, as PCI-bus limitations prevent peak throughput using dictionary-based attacks for [T1]-[T4].

Readers may modify the above thresholds but for the purpose of comparison within this document, fixed values must be used. Sections below outline those specific assumptions beyond the generally applicable above.

COMPAT, FIPS

$\langle \text{version}_{scheme}\rangle || \langle \text{salt}_{user}\rangle || \langle \text{digest}\rangle$:= HMAC($\langle \text{key}_{site}\rangle$, <mixed construct>)

- <mixed construct> := $\langle \text{version}_{scheme}\rangle$ || $\langle \text{salt}_{user}\rangle$ || $\langle \text{pw}_{user}\rangle$
- HMAC := HMAC-SHA-256
- key_{site} := PSMKeyTool(PRF-HMAC-SHA-256()):32B;
- version_{scheme} := integer (4B)
- salt_{user} := PRNG-SHA-1():32B | PRNG-FIPS186-2():32B;
- pw_{user} := <governed by password fitness>

Brief Explanation

For compatibility purposes, we base this construction on simple HMAC operation. Classic scheme components such as user-specific salt and a site-keyed HMAC provide resistance against brute force attacks. The legacy Java5 environment provides all chosen primitives and each has achieved FIPS certification (in support of those for whom this is an important requirement).

The scheme's inclusion of versioning allows initial support for in-place scheme rotation. That is, password validation routines can take into account with what version of the scheme a stored password was protected and thus appropriately apply the correct validation routine. Other aspects (fixed-size fields, delimiters between variable length fields, and inclusion of the user's name) provide protection against other modeled attacks (such as [T5.AVR06-07]).

Considerations

This section describes the solution against this document's basic and, when necessary, its subtle cryptanalytic concerns.

- ATK-1.1 - Resist "chosen plain-text" attacks
  - Applying the scheme server-side does not reveal its detail
  - The scheme's security is based on the size/complexity of its inputs (salt_{user} and pw_{user}) as well as the search space of the key_{site}
- ATK-1.2 - Resist Brute force attacks
  - Attack on individual
    - O(pw_{user}<length, complexity>* key_{site}<size>)
    - Dictionary, >= $O(2^{256}*2^{21}) = 2^{277}$
    - Exhaustive, low fitness >= $O(2^{256}*2^{186}) = 2^{442}$
    - Salt disallows precomputed table, demands "on-line" attack, does not affect speed
  - Attack on population
    - O(pw_{user}<length, complexity>* salt_{user} * key_{site}<size>)
    - Exhaustive, low fitness >= $O(2^{256}*2^{186}*2^{256}) = 2^{698}$
    - Salt disallows precomputed table, demands "on-line" attack, forces attacker to brute force each DB entry individually.
- ATK-1.3 - Resist Entropy DoS

    - No randomness required for password verification
    - 32B randomness upon password update
    - 32B randomness
- ATK-1.4 - Prevent bulk exfiltration
    - N/A
    - **IMPL NOTE: key_{site} not to be stored in DB
- ATK-1.5 - Resist identifying identical credentials
    - $salt_{user}$ provides this feature
- ATK-1.6 - Prevent Threat from producing <protected>(PW) knowing pw_{user} and $salt_{user}$
    - key_{site} provides this feature
- ATK-1.7 - Prevent exfiltration of secrets
    - See ATK-1.4
- ATK-1.8 - Prevent side-channel / timing attacks
    - N/A to this solution but:
        - **IMPL NOTE: digest comparison explicitly prevents this
        - **IMPL NOTE: provide option for delay on incorrect guess
- ATK-1.9 - Prevent extension, or similar
    - HMAC construction prevents length extension attacks
    - Fixed size salt prevents variation on this attack
    - Delimiter between username and password prevent attacks using construction on these two constructs
- SCC-1.1 - Prevent multiple encryption problems
    - N/A w/ this construction
- SCC-1.2 - Prevent common key problems
    - N/A w/ this construction
- SCC-1.3 - Prevent plain text / material leakage through construct
    - HMAC i_pad o_pad construct prevents this
- SCC-1.4 - 32B meets minimum requirement.
- SCC-1.5 - N/A w/ HMAC construction
- SCC-1.6 - N/A w/ this construction
- SCC-1.7 - N/A w/ this construction
- ***-1.01 - Addition of an internal (server-only) user-specific GUID would prevent Threats [i.e. T1, T3] from using DB write-access to overwrite stored PW
    - Including the GUID in hash data prevents this attack
    - Prepend GUID to pw_{user} and delimit from pw_{user} with a character not allowed in a user name to ATK-1.9

The compatibility-driven FIPS solution represents a surprisingly high bar against reversing through brute force means, GPU or otherwise. This scheme resists attack from [T1 - Internet-based Threat] and [T5 - Concerted Attacker]. No known scheme intends to prevent attack against [T4 - MiB] or [T2 - MiM] when SSL has been compromised.

For [T5] and [T3 LAN-based], breaking this scheme remains intractable unless threats steal both the password database and HMAC key material. If deployment separates application server and database, attack requires two separate compromises on two different systems (DB & JEE container).

Limitations

1. The implementation intends to separate this scheme's HMAC key and source material, confining this material to a protected store on the application server and protecting it in memory while running.

2. Adopters must assure [T3] has no access to HMAC key material. Single-host deployment (collocated DB and JEE container) may help defeat this compartmentalization and separation of privilege.

   Using distinct systems make achieving compartmentalization and separation of privilege between application server and database easiest. Smaller deployments that combine application server and database can achieve a similar level of security by using a Mandator Access Control (MAC) scheme on the deployed platform. For instance, *seLinux* under linux would allow complete separation of database and application server administration. Used in combination with hardening (PAX/GRSecurity) makes defeating MAC compartments infeasible for [T3].

   Once the threat completes theft of both key material and the password database, reversing (an individual or population of) passwords backs-out to O(SHA256(<$\text{salt}_{\text{user}}$>*<$\text{pw}_{\text{user}}$>)).

3. Security posture depends on proper protection of HMAC key material (key_{site}) both in use and at rest. In practice, this has proven challenging to developers. PSMKeyTool seeks to solve this problem.

4. This scheme provides no rotation or password database update without user interaction or other out-of-band credential reset.

5. If the implementor uses a user GUID in the hash, that demands that hash occur in the application server, not the database, so to compartmentalize it against DB-based attack. Password updating/checking logic must take this into account. For instance, GUID changes require re-computation and storage of the <protected>(PW).

   User GUID may *not* be transmitted to the client under this scheme. Likewise, username may *not* be used as GUID because compromise would yield the threat both username and password--sometimes the complete set of credentials required for login.

6. Including a version adds complexity. Ideally, versioning would indicate HMAC key (key_{site}) rotation, as well as whatever future changes to the <mixed construct> or other algorithm properties occur. Without more detailed design, no implementation guidance exists for handling retention of key material and functionality to validate previous <protected>(PW) versions.

Discussion of Alternatives

During the a period of review and comment on this scheme we rejected specific alternatives explicitly for various reasons. This section covers these alternatives in a Q&A format.

1. “Can I just use the users name as a salt?”

   No. Salts, by definition, must be stored in plaintext, prepended to the protected password in its protected form. The stored form is:

   ```
   <salt> || <protect>(<salt> || <password>)
   ```

   In the event of bulk PW theft having used the user’s name as a salt gives the Threat both credentials (username + password) once the password is reversed. Such a database also allows the Threat to target a particular individual’s credentials for reversal.

2. “Given Question #1’s answer, what if I use the username without prepending it to the protected form when stored?”

   This appears to “work” functionally. In such a scheme the username acts as a non-cryptographically secure nonce. The scheme provides deduplication of ciphertext in the case of identical passwords if the system demands unique user names.

   Threats aware of this scheme can conduct targeted attacks on a selected user with the same burden of reversibility as with a salted scheme. They may be able to acquire a user list from the same database or a related directory. We reject ‘extra security’ provided by this scheme based on design goal (*See KCS-1.1 - paraphrasing, “No security through obscurity”*). Additionally, we reject the username as a high-quality nonce-word on grounds that 1) it is not used only once and 2) it does not represent sufficient entropy (*See SCC-1.1*). Instead we rely on the HMAC key to provide attack resistance without demanding the approach remain secret.

3. “Why not just use an adaptive hash?”

   Indeed. We know of no implementation of such adaptive primitive in the legacy Java5 JCE, or a JCA-compliant FIPS-certified alternative with suitable Enterprise-class support.

   The reader could substitute such an adaptive hash for the HMAC() function specified by this “COMPAT, FIPS” section if unconcerned about legacy platforms, FIPS certification, and enterprise support. This document will not cover detail of swapping the HMAC() in this section In fact, we have seen organizations use the following construct to this end:

<salt$_{user}$>||<digest> := HMAC(<key$_{site}$>, <mixed construct>)

- <mixed construct> := <salt$_{user}$> || PBKDF2(HMAC-SHA1, <salt$_{user}$>, pw, c=)

It should go without explanation we do ***NOT*** recommend the reader "roll their own" in substitution. The construction above leaves unanswered questions: are two salts necessary or just one? Should I hash then HMAC and Hash or the reverse (as listed)? And so forth. Addressing these questions falls beyond the scope of this document. This document describes the *adaptive* option as its own solution.

4. "Shouldn't I use SHA-512?"

   Go ahead. Output blocks will grow by 32B (256 bits) from 68B to total 100B. Functionally, throughput may slow by around 20%.

5. "Key management seems like a pain. Can't I just use a one-way function and split the digest text to make brute force attack impossible?"

   Actually, no. Because one-way functions possess the *diffusion* property comparing even a single uniquely salted password to a large population of digest texts (>=2,000,000) fails to produce collisions. That is, if a threat were to acquire a large database containing only a chunk of salted digests they could expect to brute attack that partial-digest database (using dictionary or brute force attacks) in the same order as if the digests were whole.

   This collision resistance was experimentally (albeit informally) confirmed using ~2.1M uniquely 32B salted passwords protected using PBKDF2. Using a brute force search on the salted digests comparing only 20B chunks (the worst-case scenario for an attacker with access to either part of a split digest), every password attempt produced at most one (1) match. Thus, the simulated threat was able to uniquely identify the correct password corresponding to a protected password in each and every case without requiring the full digest text.

Reversible

<version$_{scheme}$>||<ciphertext> := ENC(<wrapper key$_{site}$>, <mixed digest>)

- <mixed digest> := <version$_{scheme}$> || <salt$_{user}$> || <pw digest>
- <pw digest> := ADAPT_FUNC(<salt$_{user}$>, <version$_{scheme}$> || <pw$_{user}$>, c=)
- ENC := AES-256
- ADAPT_FUNC := PBKDF2 | SCRYPT

- key_{site} := PSMKeyTool(PRF-HMAC-SHA-256()):32B;
- $\text{version}_{\text{scheme}}$:= integer (4B)
- c= := Work factor[5]
- $\text{salt}_{\text{user}}$:= PRNG-SHA-1():32B | PRNG-FIPS186-2():32B;
- pw_{user} := <governed by password fitness>

Brief Explanation

The reversible scheme combines the attack resistance of adaptive one-way function with the increased search space and brute force attack resistance of a keyed solution. Wrapping the stored form protects the user-specific salt making attack on (any) individual user the same order difficulty as a specific user. Depending on the adaptive function used, major platforms with default crypto-providers provide robust support for prescribed primitives. Optional use of scrypt in place of PBKDF2 provides best-in-class attack resistance against even concerted attackers [T5] at the expense of support and a rigorously tested implementation.

The scheme's versioning allows for in-place key and stored-form rotation in the event of key theft. Without stealing both the key and the password database the wrapper function makes stealing the database useless. Finally, stealing the wrapper key and the password database still requires brute force equivalent in difficulty to the work factor configured for use with the adaptive one-way function.

Considerations

- ATK-1.1 - Resist "chosen plain-text" attacks
  - Applying the scheme server-side does not reveal its detail
  - The scheme's security is based on the size/complexity of its inputs ($\text{salt}_{\text{user}}$ and pw_{user}) as well as the search space of the key_{site}
- ATK-1.2 - Resist Brute force attacks
  - Attack on individual
    - $O(\text{pw}_{\text{user}}$<length, complexity>* $\text{salt}_{\text{user}}$ * key_{site}<size>)
    - Dictionary, >= $O(2^{256}*2^{186}*2^{256}) = 2^{698}$
    - Exhaustive, low fitness >= $O(2^{256}*2^{186}*2^{256}) = 2^{698}$
    - Encrypted form and salt disallow precomputed table, demands "on-line" attack, does not affect speed. Moreover, encrypted form puts salt search space 'in play' for individual as well
  - Attack on population
    - $O(\text{pw}_{\text{user}}$<length, complexity>* $\text{salt}_{\text{user}}$ * key_{site}<size>)
    - Exhaustive, low fitness >= $O(2^{256}*2^{186}*2^{256}) = 2^{698}$
    - Salt disallows precomputed table, demands "on-line" attack, forces attacker to brute force each DB entry individually.
- ATK-1.3 - Resist Entropy DoS
  - No randomness required for password verification
  - 32B randomness upon password update
  - 32B randomness

[5] Tune work-factor to needs. If ADAPT_FUNC = PBKDF2, work-factor represent iteration count and may need to be six or seven figures. If ADAPT_FUNC = SCRYPT, developers must specify work factor in terms of both CPU and memory (N= replaces c=). Developers must also specify a parallelization resistance (p=) with SCRYPT.

- ATK-1.4 - Prevent bulk exfiltration
    - N/A
    - **IMPL NOTE: key_{site} not to be stored in DB
- ATK-1.5 - Resist identifying identical credentials
    - $salt_{user}$ provides this feature, even wrapped
- ATK-1.6 - Prevent Threat from producing <protected>(PW) knowing pw_{user} and $salt_{user}$
    - Wrapper encryption protects protected passwords against this
    - key_{site} provides this feature
- ATK-1.7 - Prevent exfiltration of secrets
    - See ATK-1.4
- ATK-1.8 - Prevent side-channel / timing attacks
    - N/A to this solution but:
        - **IMPL NOTE: digest comparison explicitly prevents this
        - **IMPL NOTE: provide option for delay on incorrect guess
- ATK-1.9 - Prevent extension, or similar
    - N/A for wrapper encryption
    - Adaptive one-way function construction prevents length extension attacks
    - Fixed size salt prevents variation on this attack
    - Delimiter between username and password prevent attacks using construction on these two constructs
- SCC-1.1 - Prevent multiple encryption problems
    - N/A w/ this construction
- SCC-1.2 - Prevent common key problems
    - N/A w/ this construction
- SCC-1.3 - Prevent plain text / material leakage through construct
    - Adaptive one-way functions prevent this
- SCC-1.4 -
    - 32B meets minimum requirement for adaptive hash
    - Digest meets requirement for wrapper encryption
- SCC-1.5 - N/A w/ this construction
- SCC-1.6 - N/A w/ this construction
- SCC-1.7 - N/A w/ this construction
- ***-1.01 - Addition of an internal (server-only) user-specific GUID would prevent Threats [i.e. T1, T3] from using DB write-access to overwrite stored PW
    - Including the GUID in one-way function data prevents this attack
    - Prepend GUID to pw_{user} and delimit from pw_{user} with a character not allowed in a user name to ATK-1.9

Selecting an appropriate work factor remains an important consideration of this solution, as it prescribes an adaptive one-way function. Adopters must tune the work factor and (in the case of SCRYPT parallelization resistance). Tuning must balance the burden placed on password validation by the defender with attack resistance in case of password database and key theft.

Limitations

1. The implementation intends to separate this scheme's wrapper encryption key and source material, confining this material to a protected store on the application server and protecting it in memory while running.

2. Adopters must assure [T3] has no access to wrapper encryption key material. Single-host deployment (collocated DB and JEE container) may help defeat this compartmentalization and separation of privilege.

   Using distinct systems make achieving compartmentalization and separation of privilege between application server and database easiest. Smaller deployments that combine application server and database can achieve a similar level of security by using a Mandator Access Control (MAC) scheme on the deployed platform. For instance, seLinux under linux would allow complete separation of database and application server administration. Used in combination with hardening (PAX/GRSecurity) makes defeating MAC compartments infeasible for [T3].

   Once the threat completes theft of both key material and the password database, reversing (an individual or population of) passwords backs-out to O(ADAPT_FUNC(<salt$_{user}$>*<pw$_{user}$>*(c))). The dominant factor in this calculation depends on the specified work factor. If sufficiently low, with both PBKDF2 and SCRYPT, work will depend on lengh and complexity of the salt and password.

3. Security posture depends on proper protection of wrapper encrytion key material (key$_{site}$) both in use and at rest. In practice, this has proven challenging to developers. PSMKeyTool seeks to solve this problem.

4. While this scheme provides wrapper key rotation in the event of theft (if the password database remains intact). The scheme supports *no* update without user interaction if threats compromise the password database itself.

5. If the implementor uses a user GUID in the one-way function, that demands that function be executed in the application server, not the database, so to compartmentalize it against DB-based attack. Password updating/checking logic must take this into account. For instance, GUID changes require re-computation and storage of the <protected>(PW).

6. User GUID may not be transmitted to the client under this scheme. Likewise, username may not be used as GUID because compromise would yield the threat both username and password--sometimes the complete set of credentials required for login.

7. Including a version adds complexity. Ideally, versioning would indicate the adaptive one-way function, its work factor, or the wrapper encryption key (key$_{site}$) rotation, as well as whatever future changes to the <mixed construct> or other algorithm properties occur. Without more detailed design, no implementation guidance exists for handling retention of key material and functionality to validate previous <protected>(PW) versions.

8. As stated under this section's considerations, the work factor must be tuned in accordance with defender appetite for scale vs. threat capability. Solving this problem through selection of a sufficiently low work factor (c=1 in the case of PBKDF2) may render adaptive hash function no longer attack resistant to concerted threats [T5].

Discussion of Alternatives

1. "Wait, I thought we should never store passwords in reversible form..."

   Agreed. This scheme uses symmetric encryption to wrap the output of a one-way function. So, when a threat A) discerns the scheme, B) steals its encryption key, and C) lifts the password database, decrypting the database leaves the threat with only salted one-way function output. The wrapper's function is to introduce a key's search space (placing brute force attacks out of the realm of possibility).

2. "OK, should I use public/private crypto then?"

   In this case, that would not bolster the scheme's security posture. Because all encryption/decryption is done in a single process space (the application server) no need for the public/private key distinction exists. If the password validation took place in a different trust zone from password protection (encryption) such a scheme might begin to provide value.

3. "Shouldn't you salt/nonce the adaptive function output before encryption?"

   Schemes use a cryptographic nonce in a manner similar to a key: the nonce adds to input entropy making brute force search harder. In this scheme, the symmetric cipher's key fulfills this purpose. The $\text{salt}_{\text{user}}$ bolsters this entropy even when cipher key theft occurs.

   Schemes use salts to "de-dup" cipher-/digest-text in the case of two (2) identical plain texts. Because the inner round (the adaptive function) possesses properties of deterministic and unique output and because the function takes a $\text{salt}_{\text{user}}$ as one of its inputs, that "de-duping" has already occurred. Because adaptive functions output a fixed length and because the symmetric cipher produces deterministic and unique outputs as well, this "de-duping" is preserved through the encryption.

Adaptive One-way Functions

Perhaps obviously, a defender could choose to simply use an adaptive one-way function for password protection. Three popular schemes exist: 1) PBKDF2, 2) BCRYPT, & 3) SCRYPT. Though the underlying implementation of each varies dramatically in structure the API of each is relatively similar:

    <salt>||<digest text> := PBKDF2(<salt$_{user}$>, <pw$_{user}$>, c=)

- c := iteration count (proxy for work factor)

    <salt>||<digest text> := SCRYPT(<salt$_{user}$>, <pw$_{user}$>, N=, p= r=)

- N := work factor (CPU + Memory)
- r := internal block size
- p := resistance factor against brute force parallelism

Bcrypt varies distinctly from these two in API and structure:

    <iteration count> || <salt> || <digest text> := BCRYPT(<salt$_{user}$>, <pw$_{user}$>, c=)

- c := iteration count (proxy for work factor)

The differences in these one-way functions prove unrelated to discussion at this level. However, the reader may find the following resources on each interesting:

- Scrypt:
  - scypt by C. Percival introduction paper
  - scrypt kdf-01, Josefsson specification
- Bcrypt:
  - http://bcrypt.sourceforge.net - Source, mailing list and pointers
  - http://en.wikipedia.org/wiki/Bcrypt - High-level explanation

In each case, attacker search space is merely a function of the complexity and length of <salt$_{user}$> and <pw$_{user}$>. Each furnishes attack resistance in terms of its iteration count or work factor. For the purpose of this section, we'll refer to iteration account as a work factor for ease[6]. Attack resistance of these schemes is thus:

- Attack on individual
  - O(pw$_{user}$ <length, complexity>)
  - Dictionary, >= O(2^{21} * <work factor>) = 2^{21} * <work factor>
    - Exhaustive, low fitness >= O(2^{186} * <work factor>) = 2^{186} * <work factor>
    - Salt and work factor disallow precomputed table, demands "on-line" attack
  - Attack on population
    - O(pw$_{user}$ <length, complexity> * salt$_{user}$ * <work factor>)
    - Exhaustive, low fitness >= O(2^{256}*2^{186}*<work factor>) = 2^{442} * <work factor>
    - Salt disallows precomputed table, demands "on-line" attack, forces attacker to brute force each DB entry individually.

Brief Explanation

[6] Authors acknowledge that technically this introduced ambiguity. For instance, bcrypt uses an iteration count (64) for its encryption phase independent of the work factor governing prior key expansion. Again, this document refers to 'work factor' as a proxy for the algorithm's parameter serving as the dominant factor in scaling its use of CPU (or CPU and memory).

Adaptive one-way functions intend to 'adapt' with hardware capabilities, continuing to thwart attack. Adopters need not replace a function once chosen but instead just ratchet up its work factor. These adaptive one-way functions do not require a key (encryption or HMAC). As such adopters need not worry about protecting ancillary secrets. Additionally, no separation of privilege or compartmentalization need occur between the application server and database.

PBKDF2 enjoys broad enterprise support from default platform cryptography providers. Implementations have often been submitted to test vectors and scrutinized thoroughly.

Considerations

Bcrypt has gained broad support on a variety of platforms in several programming languages. Scrypt support exists but more sparsely than either bcrypt or PBKDF2. Its design is compelling but it is unclear whether implementations have been submitted to rigorous review or test vectors.

Limitations

1. Security posture depends on work factor. The defender suffers a burden for this work factor on the order of:

   $$load_{sec} = O\left(\frac{user_{population}}{loginRate_{peak}} \times work_{factor}\right)$$

   To break any individual password the Threat suffers a burden of:

   $$Days_{avebreak} = O\left(\frac{user_{population}}{2 \times GPU_{speedup}} \times work_{factor}\right)$$

   Breaking a particular user's password removes the '2' from the demoninator above. Likewise, using bcrypt or scrypt remove the $GPU_{speedup}$ unless the defender competes against a concerted threat [T5].

2. Adaptive one-way functions provide their adopters no recourse under attack. While the implementing organization can adjust the work factor, they must insist users reset their password as no secondary factor (such as a key) protects user credentials.

3. Work factors provide a false sense of flexibility especially in the case of PBKDF2 and scrypt. Adopters choosing to change work factors with these two schemes ***must*** keep track of scheme version information to allow for a circumstance in which active users use scheme S_{N+1} while inactive users remain on scheme S_N. Bcrypt records work factor information in its output (by prepending the work factor to the salt and one-way result). However, in every case, the scheme requires logic to handle version identification and version-specific password validation.

Some organizations avoid the invisible version problem through use of a wrapper function. Such schemes work as follows:

```
S_{N+1} = ADAPT_FUNC(<salt_{(N+1),user}>, <salt_{(N),user}> || <protected_pw>, c=YYY)
S_N = ADAPT_FUNC(<salt_{(N),user}>, <password>, c=XXX)
(where YYY >> XXX)
```

A number of techniques can be employed to tell S_{N+1} from S_N in its stored form. However, understand that ***some*** amount of additional complexity remains unavoidable to designs involving an adaptive one-way function support response to attack.

4. PBKDF2 and bcrypt provide, in their own ways, “CPU-hardness”. That is, both use a work factor to increase the amount of time required to compute a single one-way transform. Specifically:
    a. PBKDF2 work factor provides *linear* increase in CPU effort
    b. PBKDF2 resists GPU acceleration. GPUs provide only 2X performance over modern CPUs.
    c. bcrypt work factor provides *exponential* increase in CPU effort
    d. bcrypt is currently deemed resistant to GPU acceleration due to space required by blowfish key expansion.

Discussion of Alternatives

1. “Great, three functions. Which should I use?”

   There isn’t a cut-and-dry answer to this question, unfortunately, though many have strong opinions.

   Because of PBKDF2’s broad support, well-vetted nature, and time-tested implementations, it may be a good prescription for organizations reluctant to adopt the more resistant functions for one reason or another. It suffers comparatively to the other two algorithms in attack resistance.

   Scrypt remains irrefutably the best choice for absolute protection against concerted threats [T5]. Because it scales in CPU and memory usage, sufficiently-large work factors and parallelism resistance will defeat even hand-crafted brute forcing techniques leveraging FPGAs (custom hardware). However, stymying [T5] will almost undoubtedly create an unacceptable user experience / scalability proposition for the defender.

   Bcrypt, arguably, is a harder algorithm to prescribe because of its “middle child” status. Concerted threats [T5] may be able to hand-craft FPGA-based attacks on bcrypt. However, trading off resisting this threat for bcrypt’s broader support may make perfect sense to some.

2. “Having read this, I’m wondering, could I introduce a HMAC to an adaptive one-way function to achieve defense in depth?”

Yes, yes you could. This document's authors have observed this technique "in the wild". However, we suggest the reversible scheme in this case because 1) it contains the advantages of the adaptive one-way functions and 2) possesses features amenable to key maintenance, rotation, and response to key theft (see previous section).

Workflow under Attack

Regardless of the chosen engine a successful secure password storage scheme relies on the ability to 1) protect against a threat's use of stolen credentials, 2) update/rotation of the compromised scheme, and 3) update of existing active and inactive user credential data. Though this document's last section, this is singularly the most important topic. PSM's design must:

1. Protect the user's account
    a. Invalidate authN 'shortcuts' allowing login without 2nd factors or secret questions
    b. Disallow changes to account (secret questions, out of band exchange channel setup/selection, etc.)
2. Integrate new scheme
    a. Hmac(), adaptive one-way function, reversible, etc.
    b. Include version information stored with digest
    c. Potentially include 'tainted' bit until user resets credentials
3. Wrap/replace legacy scheme: (incrementally when user logs in--#4)
    a. version||salt_{new}||protected = schemenew(salt_{old}, $\text{digest}_{\text{existing}}$) –or–
    b. For reversible scheme: rotate key, version number
4. When user logs in:
    a. Validate credentials based on version (old, new); if old demand 2nd factor or secret answers
    b. Prompt user for PW change, apologize, & conduct OOB confirmation
    c. Convert stored PWs as users successfully log in

Note that existing password storage schemes do not implement the above workflow. Invariably, some functionality described above will require tight integration with AuthN frameworks. For PSM's first iteration, stub functions will be provided to support such integration but without taking the task of integration on.

Keystore

TBD in next document revision.

Rotation

TBD in next document revision.

Conclusion

This document seeks to educate its reader in designing an attack resistant secure password storage system. Rather than proposing a single solution the document provides a generic threat model, including canonical threats, their capabilities, and likely attack vectors.

With a fixed threat model, this document presents its reader with a set of designs, tailored for particular constraints and goals. It aims to sufficiently equip the reader so that they may finalize a design tailored for their situation and--more importantly--straightforwardly and successfully articulate their chosen design’s tradeoffs in the company of architects and developers.