BreakingVault

SAP DataVault Security Storage vulnerabilities

Technical details

Author:

Fernando Russ (fruss@onapsis.com) – Sr. Researcher

Abstract:

This document describes a series of vulnerabilities found at the SAP DataVault secure storage.
The SAP Mobile Platform 3.0 has an API called DataVault, which is used to securely store data on mobile devices. As described by SAP AG [...]"The DataVault APIs provide a secure way to persist and encrypt data on the device. The data vault uses AES-256 symmetric encryption of all its contents. The cryptographic key is computed as a hash of the passcode provided and a "salt" value that can be supplied by the device application developer, or automatically generated through the API[..]"[1]

1 http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.infocenter.dc01930.0233/doc/html/aba1321994154298.html

Overview

This document covers and describes the vulnerabilities (CVE numbers pending to be assigned) enunciated above, all of them are by products of original security research performed at the Onapsis Research Labs between August 2014 and March 2015.

- Keystream recovery for secure storage
- Predictable encryption password for configuration values
- Predictable default encryption password for secure storage
- Use of incorrect cryptographic primitive to check password validity

Keystream recovery for secure storage

Due the incorrect selection of the cryptographic parameter IV used for the AES-256 algorithm in Cipher Feedback (CFB) operation mode it is possible to recover the keystream for the first 16 bytes of encrypted data of every key and every value in the DataVault. As a result, it is possible to recover the first 16 bytes (first block) of the plain text corresponding to an encrypted piece of data, reverting the encryption process of some values inside the DataVault without needing the original secret key.

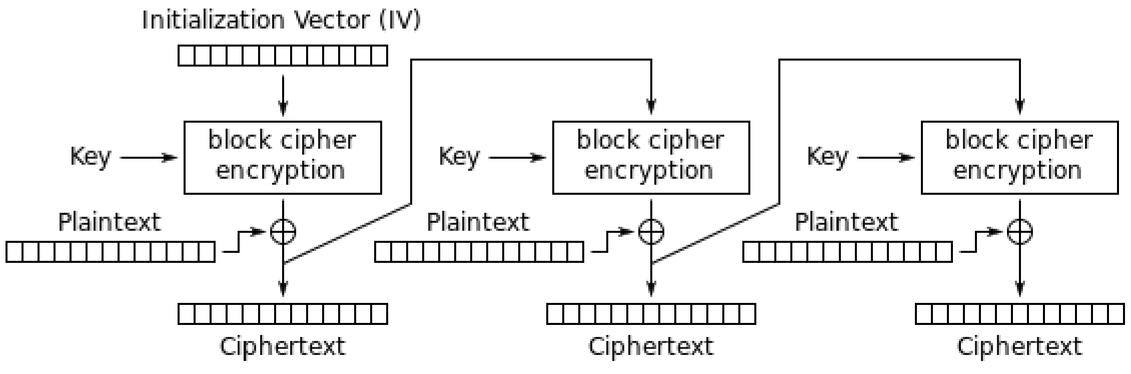
The encryption schema used by the SAP DataVault is based on the AES-256 algorithm using the Cipher Feedback (CFB) operation mode. As described in NIST SP800-38a[2] document this mode of operation needs a specific parameter IV to be **unpredictable** (page 11). It also notes that the IV value has **not to be reused** between different executions of the encryption function using the same key (page 20).

Specifically, the SAP DataVault usage of AES/CFB violates the afore mentioned recommendations having a fixed IV. As a result an attacker can easily recover the keystream for the first 16 bytes (first block) and then recover the plain text from the encrypted data. Moreover, since the IV is fixed, it is possible to ensure the presence of several known plain text values, which can be used to optimize the plain text recovery as described bellow.

Furthermore, due the lack of cryptographic integrity mechanisms in the SAP DataVault an attacker recovering this keystream has the possibility of re encrypting (or modifying in practical terms) any encrypted value shorter than 16 bytes.

Exploitation of a non-unique IV value in the mode of operation CFB

Having the Cipher Feedback (CFB) mode of operation described as,



Cipher Feedback (CFB) mode encryption

(http://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_feedback_.28CFB.29

2http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

The encryption process of the first 16 bytes of a plain text (**P**) and a secret **key** with an **IV**, looks like,

$C_p = AES_{key}(IV) \oplus P$

And, there is a known plain text (**KP**) which is encrypted using the same method, sharing the same secret **key** and **IV**,

$C_{kp} = AES_{key}(IV) \oplus KP$

Due the lack of precautions in the selection of the **IV** value (inside SAP DataVault it is constant) the $AES_{key}(IV)$ becomes constant, therefore recovering the keystream is reduced to a simple arithmetic operation,

$K_s = (AES_{key}(IV) \oplus KP) \oplus KP$
$K_s = AES_{key}(IV)$

Having this keystream K_s recovering the plain text **P** becomes trivial:

$P = C_p \oplus Ks$ which is equivalent to: $P = (AES_{key}(IV) \oplus P) \oplus AES_{key}(IV)$

Predictable encryption password for configuration values

The SAP DataVault uses a special password derived from well-known values to encrypt some configuration values like the count of invalid attempts to unlock a secure store.

This password is a composition of the VaultId value which is available in plain text form in the secure store container, and a fixed value. Also, the salt used is a fixed value. Both values are statically defined by the SAP DataVault implementation, not depending on the installation or the usage.

An attacker needs access to the file containing the SAP DataVault encrypted data, this could be achieved gaining physical access to the mobile device, or using some any other android related vulnerability which lets access a specific file in the device.

Leveraging this vulnerability an attacker could read and modify sensitive configuration values in the SAP DataVault like the count of invalid attempts to unlock the secure store.

Predictable default encryption password for secure storage

The SAP DataVault has a special mechanism to generate default set of credentials if no password/salt is supplied during the creation of the secure storage.

In this mode of operation the password/salt is derived from a combination of fixed value and the VaultID belonging to the secure storage.

An attacker needs access to the file containing the SAP DataVault encrypted data, this could be achieved gaining physical access to the mobile device, or using some any other android related vulnerability which lets access a specific file in the device.

Leveraging this vulnerability an attacker could decrypt the full content of a secure storage.
Weak key stretching algorithm used for passwords calculation

SAP DataVault applies a process of key stretching to the password used for the secure store database, this process is based in applying a single round of SHA256 over a concatenation of the password and the salt chosen by the user.

This method is effective to ensure the correct size of the final key (256 bits), but doesn't ensure any kind of key stretching, even the final key strength will be mandated by the SHA256 algorithm being deterministic and fast representing no obstacle for a determined attacker trying to bruteforce the original value.

Furthermore, the key stretching algorithm used by the SAP DataVault is

FinalKey = SHA256(key || salt)   (where || represents 'concatenation')

“key || salt” can be seen as an opaque constant string as the concatenation operation blends the limits of both values (key and salt), suppose an key **K** of 5 bytes and a salt **S** of 3 bytes,

K_1= 12345, S_1=123, then FinalKey_1 = SHA256($K_1 \| S_1$) $\rightarrow$ SHA256(123451234)

also, suppose that **K** is 7 bytes and **S** is 1 byte

K_2= 1234567 S_2=8 then FinalKey_2 = SHA256($K_2 \| S_2$) $\rightarrow$ SHA256(12345678)

as a result, the use of this kind of key stretching algorithm combined with a user-chosen salt value don't prevent a determined attacker to stage some kind of dictionary or rainbow table attacks over this.

Use of incorrect cryptographic primitive to check password validity

SAP DataVault implements a password checking strategy based on the encryption of a well-known value with a symmetric cipher algorithm, using the actual password/salt to obtain a checking blob. To verify if the supplied password/salt is the correct one, the checking blob value is decrypted and compared against the previously used well-known value.

In particular, SAP DataVault implements this password checking schema using AES-256 in OFB mode of operation over a well-known check value, plus the whole schema represents a known-plain text scenario, where the only unknown is the key itself.

Being an algorithm extremely fast with modern hardware, plus the possibility of verifying the correct decryption against the well-known check value, stage a perfect scenario for a dictionary/brute force attack.

Based in the previous description, an dedicated attacker can stage a reasonable dictionary/brute force attack against the stored password.