SafeNet, Inc.
4690 Millennium Drive
Belcamp, MD 21017 USA
Tel.: +1 410 931 7500
www.safenet-inc.com

Attacking and Fixing PKCS#11 Security Tokens

A Response by SafeNet Inc.

In their paper "Attacking and Fixing PKCS#11 Security Tokens", Bortolozzo et al., demonstrate weaknesses in the PKCS#11 standards. These weaknesses relate to the ability to extract sensitive keys by using the Wrap functionality in an unintended way. In addition to a theoretical analysis of the standard that reveals new, previously unreported attacks, an important part of their work was to analyze actual security tokens to see if real implementations are also vulnerable to these attacks. Indeed, they discovered that this was the case (and that some products are even vulnerable to attacks that are expressly forbidden by the PKCS#11 standard).

One important point to realize when reading this work is that attacks a1 and a2 as described by Bortolozzo et al, are weaknesses in the design of the PKCS#11 standard, and not necessarily flaws introduced by poor implementation of the standard. Furthermore, in real-world industry solutions, accurate compliance with standards is of great importance, and enables different products that are compliant with the standard, to work together. In the specific case under consideration, compliance of a security token with PKCS#11 enables any application that is also compliant with PKCS#11 to use the security token.

The above introduces a serious dilemma. In this case the standard in question has a weakness. At the same time, failure to remain compliant can prevent the adoption of a company's product. Furthermore, lack of compliance will not increase security overall because products that require compliance will still have to use a token that suffers from the design flaw.

While there are some inherent weaknesses in the specification, it is possible to address them in some fairly straightforward ways using best practices and tools that are already available in PKCS #11.

Let's look at three approaches that may be used to address these weaknesses:

1. **Blocking sensitive key export completely:** By patching the implementation of PKCS#11 and adding a new configuration flag to the token to enforce a stricter policy, we can disable the ability to export sensitive keys.
   - If the flag is set, wrapping is completely blocked for all symmetric keys that are determined sensitive. In this case, we lose functionally and may prevent compatibility with applications that require it. However, we ensure that any organization that does not intentionally require wrapping of symmetric keys is protected from attacks.
   - If the flag is not set, full compliance with PKCS#11 is maintained, with the unfortunate possibility that attacks a1 and a2 can be carried out.

   SafeNet has provided customers with the ability to define their tokens so that key wrapping is blocked as described in point 1 above. This addresses the concern of those who do not need this specific functionality. Fortunately we have found that this does not affect the usability of our tokens and thus setting this flag completely blocks the attacks described by the authors.

2. **Use of the TRUSTED Attribute:** The PKCS#11 2.20 API specification enables a key to be defined as TRUSTED. It is important to stress that a key can only be defined as TRUSTED upon token initialization (which is assumed to take place in a secure environment) or by the token administrator. Furthermore, in the current PKCS#11 2.20 specification, an additional attribute (of

SafeNet, Inc.
4690 Millennium Drive
Belcamp, MD 21017 USA
Tel.: +1 410 931 7500
www.safenet-inc.com

symmetric keys) has been added. This attribute is called WRAP_WITH_TRUSTED and when set, determines that the given key can only be wrapped with a trusted key.

Using the above mechanism addresses the weaknesses discovered by Bortolozzo et al. for the following reason: If a token uses a symmetric key that needs to be wrapped and exported, a trusted wrapping key is first installed on the token. After that, the WRAP_WITH_TRUSTED flag is set for any symmetric key that needs to be wrapped. Since a user (without administrator rights) cannot import or generate a trusted key on the token, he or she is unable to obtain an unwrapped version of the key. We stress that the trusted key must be defined so that it cannot be used for decryption. Otherwise, the symmetric key that is wrapped with the trusted key could be decrypted by the user.

In summary, when an application wishes to wrap and export a symmetric key, the following approach should be taken:

1) A wrapping key K1 must be set with the following attributes:
    a. CKA_TRUSTED is set to CKA_TRUE (note that this can only be set by an initialization application or the SO)
    b. CKA_WRAP is set to CKA_TRUE
    c. CKA_DECRYPT is set to CKA_FALSE
    d. CKA_SENSITIVE is set to CKA_TRUE
    e. CKA_ALWAYS_SENSITIVE is set to CKA_TRUE
2) The symmetric key K2 that needs to be wrapped and exported with K1 should be set with the following attributes:
    a. CKA_WRAP_WITH_TRUSTED is set to CKA_TRUE (Note that the default is CKA_FALSE, so this must be explicitly set)
    b. CKA_SENSITIVE is set to CKA_TRUE
    c. CKA_ALWAYS_SENSITIVE is set to CKA_TRUE
    d. CKA_EXTRACTABLE is set to CKA_TRUE

Given the above, it is possible to wrap key K2 with K1 and then export it. Unwrapping can only take place by someone familiar with K1. In addition, unless an additional TRUSTED key is added to the token, no other key can be used to wrap K1.

(While we have described only those key properties that are relevant to the issues mentioned in this paper, all of the other key properties should of course be correctly set.)

3. **Follow best practices when setting sensitive key attributes:** If you have secret or private keys that are particularly sensitive and you want to prevent them from being wrapped off, they can be generated with their template attributes: CKA_SENSITIVE and CKA_PRIVATE set to True and CKA_EXTRACTABLE and CKA_MODIFIABLE both set to False. This way, the keys are only accessible by a user who is logged in, and key values cannot be read by anyone. Also, the keys cannot be wrapped off and the attribute values cannot be changed at some later time to invalidate the original settings.

We conclude by remarking that the weaknesses discussed above, which were discovered by Bortolozzo et al., while inherent to the PKCS #11 specification, can nonetheless be overcome provided that some simple usage rules are followed. Assuming that the above described steps, or similar ones, are followed, applications can be written to be secure.