Random Oracle

Building and breaking systems

On Safenet HSM key-extraction vulnerability CVE-2015-5464 (part I)

i
8 Votes

This series of posts is provides a more in-depth explanation of the key-extraction vulnerability we discovered and reported to Safenet (https://blog.gemini.com/your-bitcoin-wallet-may-be-at-risk-safenet-hsm-key-extraction-vulnerability/), designated as CVE-2015-5464 (http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2015-5464).

PKCS11

Safenet HSMs (http://www.safenet-inc.com/data-encryption/hardware-security-modules-hsms/) are closely based on the PKCS#11 (https://en.wikipedia.org/wiki/PKCS_11) specification. This is a de facto standard designed to promote interoperability between cryptographic hardware by providing a consistent software interface. Imagine how difficult it would be to write a cryptographic application such as Bitcoin wallet to work with external hardware if each device required a different API for signing a Bitcoin transaction. Certainly at low-level differences between devices are apparent: some connect over USB while others are addressed over TCP/IP, each device typically requires different device driver much like brands of printers do. Instead PKCS11 seeks to provide a higher-level point where these differences can be abstracted behind a unified API, with a vendor-provided *PKCS#11 module* translating each function into the appropriate commands native to that brand of hardware.

PKCS#11 is a very complex standard with dozens of APIs and wide-range of cryptographic operations, called “mechanisms” for everything from encryption to random number generation. Safenet vulnerability involves the key derivation mechanisms. These are used to create a cryptographic key as a function of another key. For example BIP-32 for Bitcoin proposes the notion of hierarchical-deterministic wallets where a family of Bitcoin addresses are derived from a single “seed” secret. Designed properly, key-derivation provides such an amplification effect while protecting the primary secret.Even if a derived key is compromised, the damage is limited. One can not work their way back to the seed. But when designed *improperly*, the derived key has a simple relationship to the original secret and leaks information about it.

Some options are better left unimplemented

That turns out to be the problem with several of the key-derivation mechanisms defined in PKCS#11 and implemented by Safenet. (To give a flavor of what is supported, here is the list of options (https://randomoracle.files.wordpress.com/2015/07/extractkeyfromkey.png) presented by the demonstration utility ckdemo shipped as part of Safenet client.) Many of these are sound. A few are problematic, with varying consequences. For example the ability to toggle secret-key bits using XOR and perform operations with the result leads to exploitable conditions for certain algorithms.

Related-key cryptanalysis is the specific branch specializing in these attacks. It turns out that for Safenet HSMs, we do not need to dig very deep into cryptanalytic results. There are at least two mechanisms that are easy to exploit and work generically against a wide-class of algorithms: extract-key-from-key and XOR-base-and-data.

Slicing-and-dicing secrets

Extract-key-from-key is defined in section 6.27.7 of PKCS#11 standard version 2.30 (http://www.cryptsoft.com/pkcs11doc/STANDARD/pkcs-11v2-30m1-d7.pdf). It may as well have been renamed "extract-substring" as the analog of standard operation on strings. This derivation scheme creates a new key by taking a contiguous sequence of bits at desired offset and length from an existing key. Here is an example of this in action with ckdemo utility (http://cloudhsm-safenet-docs.s3.amazonaws.com/007-011136-002_lunasa_5-1_webhelp_rev-a/startpage_Left.htm#CSHID=reference%2Fckdemo_menu.htm%7CStartTopic=Content%2Freference%2Fckdemo_menu.htm%7CSkinName=SafeNet) provided by Safenet.

We start out with an existing 256-bit AES key with handle #37. Here are its PKCS #11 attributes:

(https://randomoracle.files.wordpress.com/2015/07/originalkeyproperties.png)
PKCS #11 attributes of original AES key

Note CKA_VALUE_LEN attribute is 0x20 in hex, corresponding to 32 bytes as expected for 256-bit AES. Because the object is sensitive, those bytes comprising the key can not be displayed. But we can use key-derivation mechanism to extract a two-byte subkey from the original. We pick extract-key-from-key mechanism, start at the most-significant bit (ckdemo starts indexing bit-positions at 1 instead of 0) and extract 2 bytes:

(https://randomoracle.files.wordpress.com/2015/07/extractkeyfromkey.png)
Using extract-key-from-key derivation

Now we look at attributes of the derived key. In particular note that its length is reported as 2 bytes:

(https://randomoracle.files.wordpress.com/2015/07/derivedkeyproperties.png)
PKCS #11 attributes of derived key

So what can we do with this resulting two-byte key, which is not going to be very difficult to brute-force?

Safenet supports HMAC with arbitrary sized keys so we can HMAC a chosen message:

(https://randomoracle.files.wordpress.com/2015/07/hmac_chosen_message.png)
HMAC chosen message using derived key

Given this primitive, the attack is straightforward: brute-force the short key by trying all possibilities against known message/HMAC pairs. In this case we get 0x5CD3 since:

```
$ echo -n ChosenMessage | openssl dgst -sha256 -hmac `echo -en "\x5c\xd3"`
(stdin)= 1db249f0e928b3aeff345aedaa3365ea690f06f3710433fc4a063b4cfffbe930
```

That corresponds to the two most-significant bytes of the original key. Now we can iterate: derive another short-key at different offset (say bits 17 through 32), brute-force that using a chosen message attack, repeat until all key bytes are recovered. Fully automated, this requires a couple of seconds with Luna G5, much less time with the more powerful SA7000 used in CloudHSM. Main trade-off is available computing power to brute-force key fragments offline. Given more resources, larger fragments of multiple contiguous bytes can be recovered at a time, necessitating fewer key derivation and HMAC operations. (Also since we have a chosen-plaintext attack with HMAC input that we control, there are time-space tradeoffs to speed up key recovery by building look-up tables ahead of time.)

Surprisingly this works not only against symmetric keys such as AES or generic HMAC secrets but also against elliptic-curve private keys (RSA, plain DSA and Diffie-Hellman were not affected.) This is an implementation quirk: these mechanisms are typically intended for symmetric-keys only. For elliptic-curve keys, the byte array being truncated is the secret scalar part of the key. For example the "secret" component for a Bitcoin ECDSA key is a discrete logarithm in secp256k1. Internally that discrete logarithms is just stored as 32-byte scalar value, and extract-key-from-key can be used to successively reveal chunks of that scalar value.

XOR-base-and-data suffers from a very similar problem. This operation derives a new key by XORing user-chosen data with original secret key. While there are cryptographic attacks exploiting that against specific algorithms such as 3DES, a design choice made by Safenet leads to simpler key recovery attack that works identically against any algorithm: when the size of data is less than size of the key, result is truncated to data size. XORing 256-bit AES key with one-byte data results in one-byte output. That provides another avenue for recovering a key incrementally: we derive new HMAC key by XORing with successively longer sequences of zero bytes, with only the last segment of new key left to brute-force at each step.

[continued in part II: exploit conditions, workarounds and mitigations (https://randomoracle.wordpress.com/safenet-part-2)]

CP

AUGUST 13, 2015 CEM PAYA

3 thoughts on "On Safenet HSM key-extraction vulnerability CVE-2015-5464 (part I)"

1. lsadehaan says:
   That is why you should only allow the minimal required attributes on your keys. You should not enable Sign for a key with Derive enabled. If you really need to derive a key for signing from a master key, then you would need to find a better method of doing the derivation instead of enabling Derive.

   REPLY DECEMBER 9, 2015 AT 16:59
2. lsadehaan says:
   I was under the impression that any key derived from a master key would have the same attributes as the master key, but I just checked the PKCS11 documentation and it only has restrictions on the sensitive and exportable attributes. This means that if a key has the derive attribute set, you can effectively create a new key with the same value that has any attributes you like (except exportable and sensitive). This means you are completely right that any key with Derive set to true would be vulnerable to these attacks.

   REPLY DECEMBER 10, 2015 AT 09:20
3. Daniel Turissini (@turissd) says:
   This a a general failure of the NIST (and other federal) testing/ evaluation programs in a PC environment that values the appearance of "equality" above the equally strict application and transparency of basic security requirements. All evaluated products are not equal, however, watering down of minimal requirements falsely cause the user community to assume one product is just as good as the other – "it's approved". For example, there are other FIPS 140-2 approved products that only support FIPS-approved KDFs, which use hashing and always use the entire secret, not parts of the secret as described in the vulnerability. SPYRUS is a prime example of a product that takes its position as a leading edge FIPS-approved product vendor seriously – here is a good read: http://www.spyrus.com/whats-a-pkcs-11-attack-and-how-do-you-prevent-it/

   Take away, buyer beware. Look beyond the basic stamp of approval.

   REPLY JULY 1, 2016 AT 10:53

BLOG AT WORDPRESS.COM.