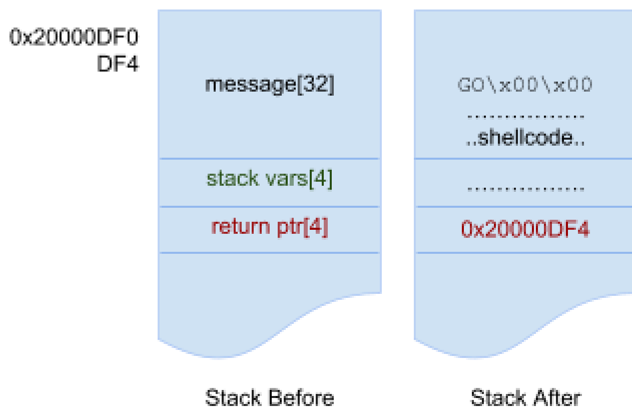
Anonymous Elephants · Tufts University · MITRE eCTF 2018

Vladimir Porokhin, Geoffrey Keating, Spencer Perry, Juliana Furgala, Richard Preston, Robert Goodfellow, Tom Hebb

In the modern world, software and embedded systems are charged with the difficult task of keeping sensitive information secure. Analyzing and breaking these systems can teach designers key lessons in secure practices. In this report, we present a two-part attack on one of the systems that first compromises a card with physical access, then effectively nullifies the security of a skimmed card as well.

The system discussed uses a symmetric encryption key and initialization vector (IV) shared between the card and server for securing communications over a network. The card also stores a secret salt for signing the pin, while the server stores the hash of the correct signature for verification. By separating the card salt from the server, data breaches require attackers to compromise two subsystems. A typical authentication process entails having the card combine its secret salt with the user-inputted pin and a server-issued nonce. With this information encrypted, the card can prove both knowledge of its secrets and user's knowledge of the pin, without disclosing them or allowing response replayability.



For communication with the ATM, the card uses a serial protocol with variable-length messages. When first plugged in, the card receives a few spurious characters, which are rejected by a synchronization routine. Unfortunately, the function did not check the length of a message against the size of the 32-byte buffer meant to hold it, allowing an attacker to write up to 224 bytes of data outside its intended bounds. `syncConnection()` allocated the buffer on the stack immediately above the location where the compiler pushed its return address. Thus, this pointer could be overwritten, causing the CPU to jump to an attacker-determined address upon return. Because the memory region occupied by the stack is executable on PSoC 4200[1,2], an *arbitrary code execution attack* was possible.

This example of a *buffer overflow* would not occur if the card verified that the message had a reasonable length, or forbade variable-length inputs altogether. In general, there are multiple techniques of avoiding overflow issues besides explicit length checks when length variability is required, including marking the stack as non-executable (DEP), randomizing memory addresses (ASLR), or using unpredictable stack canary values to detect corruption[3].

For encryption, the card used *AES-256*[4] in the *cipher feedback* (CFB) mode of operation[5] with a fixed key and IV. Further, the card communicated its UUID twice, once in plaintext to allow the server to retrieve the appropriate key/IV pair, and once in encrypted state for authentication, allowing the use of different cards' credentials for two different purposes. This is a protocol vulnerability because it allows the attacker to access an account without having its key/IV pair. There is no legitimate circumstance when credentials from two different cards would be involved in one transaction, so the server could

---

[1] Cypress Semiconductor, "PSoC 4200 Family Datasheet Programmable System-on-Chip." Accessed April 14, 2018.

[2] ARM Holdings, "Cortex M0 Technical Reference Manual (rev. r0p0)," 2009, page 39. Accessed April 14, 2018.

[3] Apple Developer Library, "Avoiding Buffer Overflows And Underflows," 2017. Accessed April 14, 2018.

detect this discrepancy by making sure the two UUIDs are the same or prevent it altogether by using a single UUID.

In CFB mode, the cipher does not encrypt the plaintext directly, but rather it encrypts the IV to generate an *"output block"*[5] or *"cipherpad,"* which is combined with the *plaintext* (the message to be protected) to obtain *ciphertext* (the encrypted message) by the means of a bitwise *exclusive-OR* (xor) operation. The text is encrypted in 16-byte blocks, with each block of ciphertext being used as an IV for the next, preventing two equal plaintexts from generating the same ciphertext and leaking sensitive information. Decrypting the message entails obtaining the same cipherpad and xoring it with the ciphertext to recover the plaintext. However, an attacker with access to the plaintext and ciphertext can recover the cipherpad by xoring the two together. Since the key and IV are fixed, knowledge of the cipherpad would completely bypass the encryption. This is especially problematic for this system because the card relies on encryption alone to protect its salt. This issue is an example of a secure data exposure, the third most common vulnerability according to the OWASP Top 10 list for 2017[6].

A *known plaintext/ciphertext attack* in this case would not be possible if the card followed the recommended practice of never reusing the same IV twice. Another defense would be to use a mode where the cipher encrypts the plaintext directly, such as ECB or CBC[5], where the knowledge of one plaintext/ciphertext pair would not compromise all future encrypted messages. Alternatively, the card could irreversibly combine the pin, signature, and nonce with a hashing algorithm to prevent decryption altogether.

In the first part of the attack, we extracted the secrets of the borrowed card. We constructed a message that caused the synchronization function to gracefully return -- starting it with `GO\x00` to pass the relevant string comparison check -- placed a shellcode of our design in the message body, and made the message long enough to overwrite the function's return address to that of our code on the stack. The shellcode, written in THUMB assembly[7]**,** called the `pushMessage()` function in a loop for every 32-byte chunk of the lower 32 kB of memory, effectively dumping the card's memory to the serial interface. From the memory dump we were then able to extract all card's secrets as they were located at fixed, predictable addresses in the chip's non-volatile FLASH memory. We then programmed our own card with the same secrets and UUID, thereby cloning the borrowed card.

To complete the attack we then retrieved the pin salt of the skimmed card. From the plaintext and encrypted UUID in the skimmed recordings, we recovered the card's cipherpad for the first block by taking the xor of the two together. The card returned its salt and user's pin encrypted together in the first block, so by xoring the cipher pad with the corresponding message, we subsequently recovered the pin salt. We didn't know the skimmed card's key and IV pair, but those weren't necessary because the server allowed the use of encryption secrets from a different card. Thus, our skimmed card clone reported plaintext UUID and used the encryption key of the borrowed card, but presented the skimmed UUID in the encrypted from and used the extracted salt to sign the pin, thereby gaining read and write access to the skimmed card's account.

As evident from this competition's results, secure system design demands high attention to detail. Although any given attack can often be blocked by a simple change in implementation, constant patching and modification may lead to additional vulnerabilities. In this case, designed protocols did not guarantee a secure exchange and lead to several critical security failures.

---

[4] Joan Daemen, Vincent Rijmen, "AES Proposal: Rijndael," 2003. Accessed April 14, 2018.
[5] NIST, "800-38A: Recommendation for Block Cipher Modes of Operation," 2001, page 16. Accessed March 7, 2018.
[6] OWASP, "Top 10 - 2017 Top Ten," Accessed April 14, 2018.
[7] ARM Limited, "ARM Architecture Reference Manual," 2005, page A6-1. Accessed March 20, 2018.