I.S.B.N : 978-84-88734-13-6

Estudio de modelado de periféricos para habilitar emulaciones de firmware embebido

Xabier Gandiaga, Urko Zurutuza, e Iñaki Garitano

Departamento de Electrónica e Informática
Escuela Politécnica Superior
Mondragon Unibertsitatea
Goiru 2, E-20500 Arrasate-Mondragón
Email: {xgandiaga,uzurutuza,igaritano}@mondragon.edu

***Resumen*—Los sistemas embebidos aumentan cada vez más en número y con ello también lo hacen los ataques dirigidos a estos. Uno de los factores clave para reducir la superficie de ataque es descubrir y corregir vulnerabilidades en el firmware embebido. El análisis dinámico es uno de los métodos más empleados para estos fines. Escalar el análisis dinámico es necesario para acelerar este proceso, lo que conlleva crear emulaciones del firmware que permitan prescindir del coste de compra de hardware. Identificamos el modelado de periféricos como problema central para habilitar dichas emulaciones. Listamos las características deseables de un proceso de modelado de periféricos, los retos a tener en cuenta y los diferentes procesos que se han utilizado para resolverlos en diferentes escenarios.**

***Index Terms*—Sistemas embebidos, emulación, análisis de firmware**

Tipo de contribución: *Investigación en desarrollo*

I. Introducción

El número de sistemas embebidos presentes en la sociedad ha aumentado debido a diversos factores, entre los que se encuentran (1) el abaratamiento de los microcontroladores, (2) la implementación de las tecnologías de conectividad y (3) el que las tecnologías pertinentes se hayan vuelto más eficientes [1]. Este aumento en dispositivos ha hecho que la cantidad de ataques dirigidos a los sistemas embebidos aumente [2]. Para hacer frente a estos ataques es necesario reducir la superficie de ataque de los sistemas embebidos, siendo clave el descubrir y arreglar vulnerabilidades en el firmware. Debido a que gran parte del firmware embebido es privado [3] el análisis dinámico, analizar el firmware durante su funcionamiento, es el método principal utilizado para la búsqueda de vulnerabilidades.

El análisis dinámico permite interactuar con el firmware en ejecución, lo cual permite controlar y establecer el estado de ejecución. El análisis dinámico requiere controlar la plataforma de ejecución para lo cual se puede utilizar el hardware originario del firmware como plataforma y adaptarlo a las herramientas de análisis a utilizar. Sin embargo, para poder escalar el análisis y acelerar la búsqueda de vulnerabilidades es necesario configurar una plataforma de emulación donde ejecutar los firmware y así prescindir del coste de adquirir hardware.

Al configurar una plataforma de emulación de un firmware, el mayor de los problemas es modelar los periféricos con los que el firmware interactúa. Para poder realizar este modelado se han creado varios procesos que hacen uso de análisis estático (analizar el firmware sin ejecutarlo), análisis dinámico, instrumentación (generar trazas de la ejecución a partir de código agregado) y hardware. No obstante, los procesos desarrollados hasta ahora no consiguen hacer frente a todos los problemas que el mercado de sistemas embebidos presenta.

Entre dichos problemas destacamos los siguientes: (1) los sistemas embebidos continúan en aumento, por lo que los procesos de modelado de periféricos tienen que acelerarse mediante escalabilidad. Para ello es necesario prescindir de hardware durante el proceso de modelado y automatizar el esfuerzo manual requerido, reduciendo el nivel técnico para aplicarlos. (2) Debido a la heterogeneidad de arquitecturas y periféricos en los sistemas embebidos, es necesaria una aplicabilidad genérica. Es decir, el que un proceso de modelado de periféricos funcione entre diferentes firmware. Esto implica funcionar en diferentes arquitecturas de procesadores y diferente acceso a información sobre el firmware (código abierto o privado, p.e.). Además, (3) los modelos creados deberían de ser transferibles a emulaciones de otros firmware para así reducir el esfuerzo en emular otros dispositivos.

Este documento resume los trabajos en la literatura sobre el modelado de periféricos con el fin de habilitar emulaciones para el análisis dinámico de firmware embebido.

II. Periféricos de sistemas embebidos

Los sistemas embebidos utilizan principalmente dos métodos para incorporar periféricos; (1) integrarlos en el microcontrolador y (2) conectarlos a través de entradas/salidas digitales o algún bus de comunicación. Esto separa los periféricos en periféricos *on-chip* (módulos de memoria, temporizadores, controladores de interrupciones, interfaces/buses de comunicación, etc.) y periféricos *off-chip* (cámaras, sensores, botones, etc.).

Al emular firmware de sistemas embebidos, algunos periféricos *on-chip* ya están implementados en plataformas de emulación. Sin embargo, otros periféricos *on-chip* y casi todos los periféricos *off-chip* carecen de estas implementaciones.

III. Modelado de periféricos

Modelar un periférico consiste en interceptar las interacciones del firmware con el periférico y proporcionar respuestas válidas al firmware para que la ejecución continúe. Un modelo completo es capaz de proporcionar el firmware con

valores correctos para cualquier llamada al periférico original y puede dirigir el firmware por todas sus líneas de ejecución (una cobertura completa). También es posible sustituir la implementación del periférico por el de otro similar, aunque esto puede causar que la ejecución difiera del original. Los procesos que realizan el modelado se pueden diferenciar en dos grupos [4]:

- **Modelado a alto nivel de abstracción:** El modelado de periféricos ocurre a nivel de código del firmware. Las partes del código del firmware que interactúan con los periféricos son reemplazados por código que evita la interacción y proporciona respuestas al firmware.
- **Modelado a bajo nivel de abstracción:** El firmware interactúa con periféricos a través de escrituras y lecturas a memoria. En este tipo de modelado, se intercepta el firmware mediante código agregado a la plataforma de emulación cuando este intenta leer una respuesta de un periférico en la memoria. Después ese código escribe las respuestas correspondientes en los registros de la memoria que el firmware va a leer.

IV. Técnicas para modelar periféricos

El modelado de periféricos se realiza aplicando diferentes técnicas de análisis estático, análisis dinámico e instrumentación sobre el firmware y las plataformas de ejecución en uso para el firmware (emulaciones, hardware o ambos a la vez).

Las técnicas principales aplicadas al firmware para modelar periféricos son las siguientes:

- **Instrumentación:** Instrumentar significa agregar código o programas que generen trazas sobre la ejecución. Estas trazas se pueden utilizar para crear modelos o como base para los *inputs* de otras técnicas. Esta instrumentación puede ocurrir a nivel de binario (instrumentación de binario mediante angr y similares), a nivel de plataforma de emulación (PANDA [5], p.e.) o a nivel de hardware.
- **Fuzzing:** El fuzzing de firmware es un método de fuerza bruta que crea *inputs* a insertar al firmware. Los fuzzer generan valores en base a normas que delimitan cómo mutar unos *input* base. Estos valores base se formulan a través de la documentación del firmware o *logs* de ejecución del firmware. El fuzzer más utilizado para modelar periféricos es el American Fuzzy Loop (AFL) [6].
- **Ejecución simbólica:** La ejecución simbólica es un método de análisis dinámico de caja blanca o *whitebox*. Un ejecutor simbólico analiza rutas de ejecución de un código seleccionadas en base a un algoritmo exploratorio. Las rutas de ejecución se representan como un árbol binario que crea dos nuevas ramas, *true* y *false*, por cada condición encontrada en el código. Los valores que guían la ejecución por las ramas se guardan como valores simbólicos. Estos valores simbólicos se pueden resolver para obtener un valor concreto que guíe la ejecución hasta un estado específico del código. Las herramientas de ejecución simbólica más utilizadas para modelar periféricos son angr [7] y S2E [8].
- **Ejecución concólica:** La ejecución concólica es una mezcla de una ejecución simbólica y una ejecución de valores concretos. Se utiliza para evitar problemas que la ejecución simbólica puede llegar a tener en estados complejos del código (explosión de rutas, tiempos largos para resolver valores simbólicos, etc.). Para modelar periféricos, la ejecución concólica se basa en una plataforma de ejecución combinada con una herramienta de ejecución simbólica (QEMU [9] + angr o S2E).

V. Retos del modelado de periféricos

Los retos del modelado de periféricos listados los extraemos de los problemas a los que los procesos de modelado, presentados más adelante en el documento, hacen frente:

- **Uso de hardware:** En caso de ser necesario el uso de hardware real durante el proceso de generación del modelo, el escalado horizontal del proceso estará limitado por la cantidad de dispositivos reales disponibles.
- **Necesidad de trabajo manual:** A cuanto más trabajo manual necesite el proceso de modelado menos escalable es el proceso porque no se podrán automatizar esas partes. Una menor escalabilidad resulta en procesos más lentos. Además, el trabajo manual requiere de un conocimiento técnico mayor para aplicarse, lo cual reduce la usabilidad de la proceso en la industria.
- **Conexiones externas y control del hardware:** En caso de utilizar hardware durante el proceso para modelar los periféricos, los procesos aplicables están limitados a las conexiones y control disponibles del hardware (JTAG, serial, conexiones TCP, *stubs* de depuración insertables en el hardware, etc.).
- **Disponibilidad de herramientas de análisis:** Las herramientas de análisis de binarios más populares no soportan todas las arquitecturas disponibles, lo cual limita los procesos que los utilicen. Como ejemplo, S2E tuvo que ser re-adaptado a ARM en [10].
- **Firmware de código abierto o privativo:** En un proceso de modelado de periféricos a bajo nivel, si los firmware con los que trabajar son de código cerrado, es imposible utilizar técnicas de caja blanca de análisis dinámico sobre ellos (ejecución simbólica y ejecución concólica) para modelar periféricos. Trabajando en modelado de alto nivel, localizar el código que interactúa con los periféricos en el binario puede ser inviable por el mismo motivo.
- **Características concretas en el código:** Los procesos de modelado de periféricos a alto nivel requieren que el código utilice capas de abstracción para el control de periféricos. Estas capas de abstracción son *Hardware Abstraction Layers* (HAL) en firmware *baremetal* [11] y *Board Support Packages* (BSP) y *drivers* en firmware con sistemas operativos [12].
- **Acceso directo a memoria (DMA):** Mediante un controlador de DMA un periférico puede escribir en memoria sin interactuar con el procesador. Esto significa que durante la emulación, en aquellos casos en los cuales el periférico a modelar haga uso de DMA, el firmware puede quedarse a la espera de esta interacción. Si el modelo no lo contempla esto no ocurre y la ejecución podría verse afectada. Si se trabaja a bajo nivel, es necesario identificar patrones de acceso por DMA en el firmware para saber cuando una interacción por DMA

puede ocurrir. A alto nivel, es necesario localizar las funciones que interactúan con el controlador DMA y reemplazarlos.

- **Controladores de interrupciones:** Las interrupciones son señales lanzadas por el hardware que avisan de eventos asíncronos al firmware. Los controladores de interrupciones mantienen un listado de qué hardware puede lanzar qué interrupciones y la prioridad de estos. Una interacción con un periférico puede quedarse estancada si las interrupciones correctas no son lanzadas. Es necesario modelar qué interrupciones habilitar en cada momento e ir lanzándolos ordenadamente en modelos de bajo nivel.
- **Transferibilidad de los modelos:** Muchos procesos de modelado de periféricos modelan la interacción de periféricos *off-chip* con el procesador mediante periféricos *on-chip* que actúan como interfaces para el procesador [13]. Esto hace que los modelos creados sean modelos que combinan dos periféricos (interfaz *on-chip* y periférico *off-chip*), reduciendo su transferibilidad a firmware que utilice esa misma combinación de periféricos.
- **Cobertura del firmware:** Al utilizar un modelo para evitar una interacción con un periférico es posible que ese modelo no sea completo. Dicho modelo evitaría que un análisis sobre la emulación final obtuviese un cobertura total del firmware, pudiendo ocultar vulnerabilidades. Esto ocurre tanto trabajando en modelado a alto nivel como a bajo nivel [14]. A cuanto mayor la cobertura mayor es la fidelidad del modelo al periférico original.

Agrupándolos, destacamos los siguientes retos: (1) la escalabilidad del proceso de generación del modelo, afectado por el uso de hardware y trabajo manual, (2) la versatilidad de los procesos de modelado, afectada por la heterogeneidad del hardware, características del firmware y herramientas utilizables. Por último, (3) la calidad del modelo resultante en términos de transferibilidad del modelo y cobertura del firmware.

VI. Clasificación de procesos que habilitan emulaciones

El nivel de las emulaciones de firmware se puede clasificar en base a la cobertura del firmware que ofrecen los modelos de periféricos [15]. A su vez, los procesos para crear estos modelos se pueden clasificar en base al nivel de fidelidad que buscan para el modelo de un periférico y los retos del modelado de periféricos a los que responden. En base a esos factores los procesos de modelado se pueden separar en los siguientes grupos:

VI-A. Emulación hardware-in-the-loop

La emulación *hardware-in-the-loop* utiliza hardware para habilitar la ejecución de las emulaciones. Estos procesos no intentan modelar periféricos, sino que redirigen las llamadas a periféricos que realiza el firmware en la plataforma de emulación al periférico real. Sin embargo, están limitados en escalabilidad y aplicabilidad por estar atados a hardware. Los trabajos más relevantes son Avatar, de Zaddach, Bruno, et al. [16] el cual se utiliza como base de múltiples otros estudios ([17],[18]), Prospect, de Kammerstetter et al. [19] y Surrogates, de Koscher et al. [20]

VI-B. Modelado parcial

El modelado parcial no se preocupa por la fidelidad de la emulación final. Estos procesos (1) sustituyen periféricos no emulados por otros ya emulados y (2) responden a llamadas a periféricos con valores concretos que permitan continuar la ejecución. Estos valores se infieren de archivos de configuración en los sistemas de archivos del firmware a emular o se utilizan algunos definidos por defecto. Firmadyne, de Chen et al. [21] y FirmAE, de Kim et al. [22] sustituyen el kernel original del firmware por otro propio (reduciendo aún más la fidelidad de la emulación final) y automatizan los pasos (1) y (2), disminuyendo el nivel técnico requerido. En Ecmo, de Jiang et al. [23], se sustituyen los periféricos de forma manual con código insertado a la plataforma de emulación.

VI-C. Modelado mediante hardware

El modelado mediante hardware intenta crear modelos de periféricos de alta fidelidad que prescinden del hardware para su uso en base a información obtenida instrumentando hardware o una emulación *hardware-in-the-loop*. Los modelos creados mediante estos procesos son modelos a bajo nivel. Pretender, de Gustafson et al. [17] utiliza instrumentación sobre Avatar para obtener datos sobre la interacción de los periféricos y QEMU. Después utiliza esta información para crear modelos de periféricos basados en inteligencia artificial. Conware, de Spensky et al. [13] instrumenta dispositivos físicos que corren firmware de código abierto para obtener *logs* en los cuales basar sus modelos. Después examina el código del firmware para dividir estos modelos en dos. Un modelo para la interfaz *on-chip* y otro para el periférico *off-chip*. Estos modelos individuales son más transferibles.

VI-D. Modelado mediante trabajo manual

Los procesos en esta categoría trabajan en el modelado de periféricos de alto nivel. El trabajo que realizan se puede dividir en dos pasos principales: (1) el localizar las capas de abstracción de uso de hardware, HAL, en el código y (2) generar el código del modelo para reemplazarlos. Este código de reemplazo intenta ser lo más fiel posible al periférico original. El localizar la capa de abstracción puede ser automatizado, pero, por ahora, el código para el modelado ha sido creado manualmente en todos los trabajos. HALucinator, de Clements et al/ [11], trabaja con HALs en firmware *baremetal*. Para-rehosting, de Li et al. [24], combina el modelado a alto nivel con transferir la lógica del firmware a un programa de *userspace* de x86. Firmporter, de Xin et al. [12] trabaja con BSPs y *drivers* en firmware para RTOS de código abierto. Clements et al. [15] trabajan con firmware de RTOS de código cerrado.

VI-E. Modelado automático

En el modelado automático se crean modelos de alta fidelidad a bajo nivel mediante procesos automatizados y escalables. Las técnicas utilizadas en esta categoría son el *fuzzing*, ejecuciones simbólicas y ejecuciones concólicas. P2IM, de Feng et al. [25] crea modelos de periféricos en firmware de código cerrado mediante la clasificación de registros de memoria y un fuzzer. Mediante el fuzzer se prueban diferentes respuestas para el firmware que se escriben en direcciones de memoria de destino clasificadas manualmente. Aunque el

clasificar las direcciones de memoria es un trabajo manual, el modelado (encontrar las respuestas correctas para el firmware) es automático. DICE, también por Feng et al. [26] agrega la posibilidad de modelar periféricos que utilizan DMA en P2IM. Jetset de Johnson et al. [27], Laelaps de Cao et al. [18] y µEmu de Zhou et al. [10] utilizan ejecuciones simbólicas (Jetset) y ejecuciones concólicas (Laelaps, µEmu) para modelar los periféricos. Sin embargo, al utilizar técnicas de caja blanca, están limitados a firmware abierto. Fuzzware de Scharnowski et al. [14] utiliza una combinación de un fuzzer y ejecuciones simbólicas. Las ejecuciones simbólicas limitan los valores y las mutaciones que el fuzzer debe probar. Esto permite aumentar la cobertura del firmware en menos tiempo de ejecución.

VII. Conclusiones y líneas futuras

La emulación de firmware de sistemas embebidos presenta múltiples retos debido principalmente a la heterogeneidad en cuanto a arquitecturas y periféricos existente. Uno de los mayores retos se centra en la emulación de los periféricos, lo cual requiere la creación de un modelo o gemelo digital lo más realista posible. En este trabajo se han recogido los distintos procesos existentes para la creación de modelos de periféricos, clasificándolos en dos grupos en base a la capa de abstracción sobre la cual trabajan, a nivel de código del firmware y a nivel de plataforma de emulación. Así mismo, se han descrito los principales retos para el modelado de periféricos y los trabajos más relevantes clasificados en base a los objetivos, retos y la fidelidad de la emulación resultante.

Las líneas futuras de este ámbito incluyen: (1) automatizar las partes manuales de los procesos citados para reducir el nivel técnico que requieren en su uso, (2) extender los procesos a arquitecturas no soportadas, adaptando las herramientas de análisis usadas y (3) trabajar la transferibilidad de modelos.

Agradecimientos

Este trabajo ha sido desarrollado por el grupo de sistemas inteligentes para sistemas industriales apoyado por el Departamento de Educación, Política Lingüística y Cultura del Gobierno Vasco (IT1676-22). Ha sido parcialmente financiado por el proyecto REMEDY del Departamento de Desarrollo Económico e Infraestructuras bajo el acuerdo de subvención KK-2021/00091. Así mismo, ha sido parcialmente financiado por el proyecto VARIOT, TENtec n. 28263632 del programa Connecting Europe Facility de la Unión Europea.

Referencias

[1] F. Xia, L. T. Yang, L. Wang, and A. Vinel, "Internet of things," *International journal of communication systems*, vol. 25, no. 9, p. 1101, 2012.

[2] P.-A. Vervier and Y. Shen, "Before toasters rise up: A view into the emerging iot threat landscape," in *International Symposium on Research in Attacks, Intrusions, and Defenses*. Springer, 2018, pp. 556–576.

[3] C. Maxfield. (2019) Embedded markets study - integrating iot and advanced technology designs, application development processing environments.

[4] A. Fasano, T. Ballo, M. Muench, T. Leek, A. Bulekov, B. Dolan-Gavitt, M. Egele, A. Francillon, L. Lu, N. Gregory, D. Balzarotti, and W. Robertson, "SoK: Enabling Security Analyses of Embedded Systems via Rehosting," *ASIA CCS 2021 - Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, no. Ii, pp. 687–701, 2021.

[5] B. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, and R. Whelan, "Repeatable reverse engineering with panda," in *Proceedings of the 5th Program Protection and Reverse Engineering Workshop*, 2015, pp. 1–11.

[6] M. Zalewski. (2021). [Online]. Available: https://github.com/google/AFL

[7] S. a. A. S. U. . S. Computer Security Lab at UC Santa Barbara. (2021). [Online]. Available: https://github.com/angr/angr

[8] V. Chipounov, V. Kuznetsov, and G. Candea, "S2e: A platform for in-vivo multi-path analysis of software systems," *Acm Sigplan Notices*, vol. 46, no. 3, pp. 265–278, 2011.

[9] F. Bellard, "Qemu, a fast and portable dynamic translator." in *USENIX annual technical conference, FREENIX Track*, vol. 41, no. 46. California, USA, 2005, pp. 10–5555.

[10] W. Zhou, L. Guan, P. Liu, and Y. Zhang, "Automatic firmware emulation through invalidity-guided knowledge inference," in *USENIX Security Symposium*, 2021, pp. 2007–2024.

[11] A. A. Clements, E. Gustafson, T. Scharnowski, P. Grosen, D. Fritz, C. Kruegel, G. Vigna, S. Bagchi, and M. Payer, "{HALucinator}: Firmware re-hosting through abstraction layer emulation," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1201–1218.

[12] M. Xin, H. Wen, L. Deng, H. Li, Q. Li, and L. Sun, "Firmware re-hosting through static binary-level porting," *arXiv preprint arXiv:2107.09856*, 2021.

[13] C. Spensky, A. Machiry, N. Redini, C. Unger, G. Foster, E. Blasband, H. Okhravi, C. Kruegel, and G. Vigna, "Conware: Automated modeling of hardware peripherals," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 95–109.

[14] "Fuzzware: Using precise MMIO modeling for effective firmware fuzzing," in *31st USENIX Security Symposium (USENIX Security 22)*, Boston, MA, 2022.

[15] A. A. Clements, L. Carpenter, W. A. Moeglein, and C. Wright, "Is your firmware real or re-hosted?" in *Workshop on Binary Analysis Research (BAR)*, vol. 2021, 2021, p. 21.

[16] J. Zaddach, L. Bruno, A. Francillon, D. Balzarotti *et al.*, "Avatar: A framework to support dynamic security analysis of embedded systems' firmwares." in *NDSS*, vol. 14, 2014, pp. 1–16.

[17] E. Gustafson, M. Muench, C. Spensky, N. Redini, A. Machiry, Y. Fratantonio, D. Balzarotti, A. Francillon, Y. R. Choe, C. Kruegel *et al.*, "Toward the analysis of embedded firmware through automated re-hosting," in *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, 2019, pp. 135–150.

[18] C. Cao, L. Guan, J. Ming, and P. Liu, "Device-agnostic firmware execution is possible: A concolic execution approach for peripheral emulation," in *Annual Computer Security Applications Conference*, 2020, pp. 746–759.

[19] M. Kammerstetter, C. Platzer, and W. Kastner, "Prospect: peripheral proxying supported embedded code testing," in *Proceedings of the 9th ACM symposium on Information, computer and communications security*, 2014, pp. 329–340.

[20] K. Koscher, T. Kohno, and D. Molnar, "{SURROGATES}: Enabling {Near-Real-Time} dynamic analyses of embedded systems," in *9th USENIX Workshop on Offensive Technologies (WOOT 15)*, 2015.

[21] D. D. Chen, M. Woo, D. Brumley, and M. Egele, "Towards automated dynamic analysis for linux-based embedded firmware." in *NDSS*, vol. 1, 2016, pp. 1–1.

[22] M. Kim, D. Kim, E. Kim, S. Kim, Y. Jang, and Y. Kim, "Firmae: Towards large-scale emulation of iot firmware for dynamic analysis." New York, NY, USA: Association for Computing Machinery, 2020.

[23] M. Jiang, L. Ma, Y. Zhou, Q. Liu, C. Zhang, Z. Wang, X. Luo, L. Wu, and K. Ren, "Ecmo: Peripheral transplantation to rehost embedded linux kernels," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 734–748.

[24] W. Li, L. Guan, J. Lin, J. Shi, and F. Li, "From library portability to para-rehosting: Natively executing microcontroller software on commodity hardware," *arXiv preprint arXiv:2107.12867*, 2021.

[25] B. Feng, A. Mera, and L. Lu, "P2IM: Scalable and hardware-independent firmware testing via automatic peripheral interface modeling," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 1237–1254.

[26] A. Mera, B. Feng, L. Lu, and E. Kirda, "Dice: Automatic emulation of dma input channels for dynamic firmware analysis," in *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2021, pp. 1938–1954.

[27] E. Johnson, M. Bland, Y. Zhu, J. Mason, S. Checkoway, S. Savage, and K. Levchenko, "Jetset: Targeted firmware rehosting for embedded systems," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 321–338.