DEMO: A Framework to Test and Fuzz Wi-Fi Devices

Domien Schepers
Northeastern University
Boston, Massachusetts, USA
schepers.d@northeastern.edu

Mathy Vanhoef
New York University Abu Dhabi
Abu Dhabi, UAE
mathy.vanhoef@nyu.edu

Aanjhan Ranganathan
Northeastern University
Boston, Massachusetts, USA
aanjhan@northeastern.edu

ABSTRACT

Over the years, numerous weaknesses have been identified in the IEEE 802.11 standard and its implementations. In order to present a proof-of-concept or demonstrate their impact in practice, researchers are often required to implement entire procedures or complex features from scratch (e.g., injecting encrypted frames with customized header flags). In this paper, we present a framework that allows researchers to more easily test and fuzz any device (i.e., access points and clients). This framework enables one to, for example, test hypothesis on new weaknesses, implement proof-of-concepts, create testing suites, and automate experiments. Our framework is implemented on top of the *hostap* user space daemon, and includes a language in which complex test cases can be defined (e.g., instructions to inject a sequence of user-modified frames into the network). Notably, a test case can make use of the *hostap* control interface, providing access to built-in features (e.g., authentication procedures, retrieval of encryption keys) and allows users to create customized *hostap* extensions.

CCS CONCEPTS

• **Networks → Wireless access points, base stations and infrastructure**; *Mobile and wireless security*.

KEYWORDS

IEEE 802.11, Wi-Fi, Framework, Security, Privacy

ACM Reference Format:
Domien Schepers, Mathy Vanhoef, and Aanjhan Ranganathan. 2021. DEMO: A Framework to Test and Fuzz Wi-Fi Devices. In *14th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec '21), June 28–July 2, 2021, Abu Dhabi, United Arab Emirates.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3448300.3468261

1 INTRODUCTION

IEEE 802.11 (Wi-Fi) is an evolving set of standards providing wireless access to local area networks. Over the years, numerous weaknesses and vulnerabilities have been discovered in the wide variety of security and privacy features offered by Wi-Fi. For example, researchers identified key reinstallation attacks [9], security weaknesses in encryption protocols such as WPA-TKIP [5, 6] and the Dragonfly handshake of WPA3 [10], and shown stealthy evil-twin attacks against enterprise networks [1]. Furthermore, researchers modeled the four-way handshake to identify weaknesses [7, 11], inspected the impact of Wi-Fi Protected Setup (WPS) flaws [4], and proposed numerous methods to enhance the security of Wi-Fi networks, for example, by presenting an extensive formal analysis of the WPA2 protocol design [2]. In their experiments, researchers are often required to implement complex features or entire procedures from scratch (e.g., using Python and Scapy). For example, it may be required to inject encrypted frames on a monitor interface (requiring knowledge of cryptographic keys), set customized flags in the header (often requiring one to bypass the driver), buffer or discard specific frames, and more, all while supporting both clients and access points. Instead of implementing this functionality from scratch (e.g., using Scapy), it is often worthwhile to use existing Wi-Fi daemons and leverage their built-in functionality. A popular and well-known example for Linux-based computers is *hostap*, an open implementation available under a BSD-license. Clients commonly install user space daemon software to connect to wireless networks or to configure and run their own wireless access points.

We present a framework that is implemented on top of *hostap*, and encompasses a language to define complex test cases which can be executed automatically against any device. Our framework provides access to the *hostap* control interface, and as such test cases can leverage existing built-in functionality. Since researchers can reuse existing code, they are able to swiftly test and fuzz any device, significantly reducing their efforts compared to an implementation from scratch. Furthermore, leveraging *hostap* has noticeable benefits. For example, an access point will automatically transmit beacon frames announcing the network, a client will not need to scan for networks, and authentication algorithms can be reused allowing test cases to be used against any network configuration (e.g., enterprise networks). Additionally, since the framework leverages *hostap* it will automatically transmit acknowledgments to incoming frames. As such, frames are acknowledged by the hardware without delay, a benefit when building fuzz-testing tools.

Our framework supports researchers and enthusiasts who wish to, for example, test hypothesis on new weaknesses, implement proof-of-concepts, build fuzz-testing tools, create testing suites (e.g., test if a device is vulnerable to known attacks), or automate experiments. Recently, researchers used the concept of our framework to demonstrate a variety of fragmentation and aggregation attacks affecting all protected Wi-Fi networks [8]. Their work demonstrates the real-world usability and value of our framework. For example, the researchers released a set of test cases to evaluate if a station is vulnerable, enabling others to easily inspect their own devices [8].

2 FRAMEWORK

In this section, we present the various components that make up our framework. An overview of our framework is given in Figure 1.

WiSec '21, June 28–July 2, 2021, Abu Dhabi, United Arab Emirates

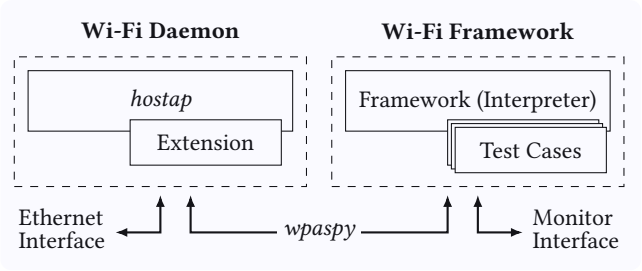


Figure 1: Overview of the Wi-Fi Daemon and Framework components and their respective communication interfaces.

Availability. Our framework and supporting documentation is publicly available on Github [1]. We provide tutorials on how to extend the control interface of *hostap* as well as our own framework and test case language, and share all test cases of our demonstration.

2.1 Implementation and Virtualization Support

Our framework is implemented in Python3 and makes use of Scapy, a popular packet manipulation program. We provide support for the most-commonly used user space daemon, *hostap*, using its latest stable release (Version 2.9). Our framework can be used with any physical hardware (e.g., Wi-Fi cards, Wi-Fi dongles) supporting virtual interfaces and the injection of raw frames (i.e., monitor mode). Additionally, our framework supports virtualized Wi-Fi radios since Linux has a kernel module which can be used to simulate an arbitrary number of WLAN radio interfaces (i.e., *mac80211_hwsim*). As such, researchers do not require hands-on access to suitable hardware, and are able to develop and run test cases through simulations preventing any real-world impact or radio interference. Furthermore, our framework includes a tool to manage Linux-based wireless network interfaces (e.g., to enable monitor mode) and supports custom *hostap* binaries and configuration files. Finally, we provide a wrapper to run a stand-alone lightweight *hostap* daemon.

2.2 Definition and Features of a Test Case

Our framework supports a Python-based language to define test cases, and can be easily extended to support user-specific needs. Fundamentally, a test case defines a number of sequential actions, which are executed upon the activation of their respective trigger. Predefined actions include *Inject*, *Receive*, *Function*, *Reconnect*, and more, offering a variety of features. Notably, *Inject* allows one to inject plaintext as well as encrypted frames onto the monitor interface. This allows one to modify the IEEE 802.11 header (e.g., to manipulate header flags), which would otherwise not be possible over the ethernet interface. Additionally, *Function* supports user-defined functions which are executed at run-time (e.g., to request connection-specific information), and *Receive* to receive frames from the ethernet or monitor interface. Predefined triggers include *Associated*, *Connected*, *Disconnected*, and more. Triggers ensure actions are executed only upon reaching a defined moment in the life-cycle of a connection, and abstract away differences between a client and an access point. For example, one can create a test where where a frame is injected only after successfully establishing a connection (i.e., after deriving fresh session keys), or when a client initiates the authentication procedure with an access point.

Ethernet and Monitor Interface. Our framework supports both the ethernet and monitor interface to receive and inject frames, enriching the test case language. For example, at times it may be convenient to use the daemon to encrypt and transmit a frame (e.g., to send a ping request), alternatively, using the monitor interface may be preferred since it gives more control (e.g., to set customized header flags). Furthermore, when injecting a plaintext frame over the virtual monitor interface, the kernel will (re)use the packet queue of the managed client to transmit the frame, meaning the injected frame will be automatically retransmitted if no acknowledgment was received. Additionally, when testing a client, the injected frame will be buffered by the Linux kernel in case the client is asleep. As such, we increase the reliability of a test case and reduce the occurrence of potential false negative results.

2.3 Extending the Daemon's Control Interfaces

User space daemons such as *hostap* have control interfaces on which commands can be executed (if enabled during compilation). Any station, that is either an access point or client, has its own control interface and respective set of commands. For example, a command on the control interface of a client station is to disconnect from the network. Using a Python-based communication wrapper named *wpaspy*, we have direct access to the control interface, and are able to issue commands. However, existing commands may be limited (e.g., they may not support the retrieval of all cryptographic keys). In order to support the definition of complex test cases in our framework, we implement extensions into the control interfaces, supporting additional commands. For example, we write a command to extract the session's temporal encryption key, since it is needed to construct encrypted frames for injection over the monitor interface. On our repository, we provide documentation on how to enable and extend the control interfaces with customized commands. Furthermore, we provide a patch file for the implementation of commonly used commands. As such our modifications are transparent, and can be applied to already-deployed systems.

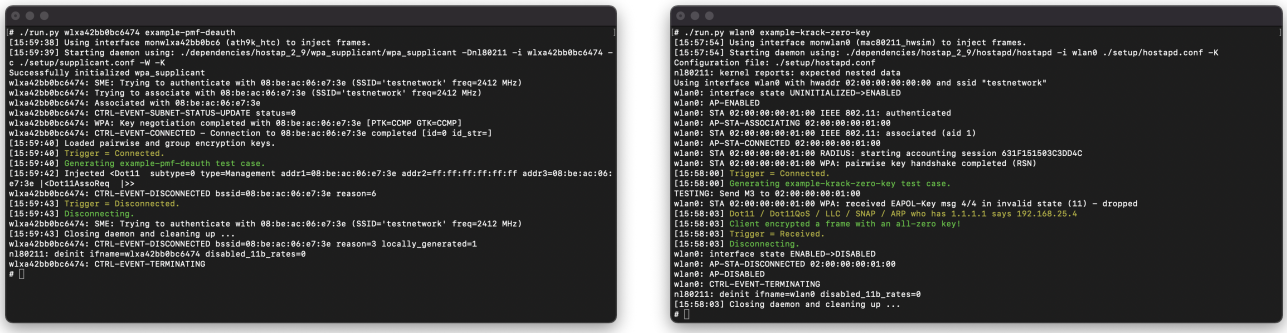
3 DEMONSTRATION

We demonstrate the strengths of our framework by creating two distinct test cases, evaluating an access point as well as a client station, thereby covering a wide range of research scenarios. First, we define a test case which evaluates if a Wi-Fi PMF-enforced access point is vulnerable to a deauthentication attack [3]. Second, we evaluate if a client is vulnerable to key reinstallation attacks [9].

3.1 Wi-Fi PMF Deauthentication Vulnerability

The IEEE 802.11w amendment standardized Wi-Fi Protected Management Frames (PMF) and provides protection mechanisms for management frames. For example, it prevents deauthentication attacks in which an adversary forcibly disconnects clients from a network. The Wi-Fi Alliance made Wi-Fi PMF part of certification programs [12], and is mandatory in modern network configurations such as WPA3. In 2019, vulnerabilities were identified in *hostap* (CVE-2019-16275) and are resolved in its upcoming v2.10 release [3].

[1] https://www.github.com/domienschepers/wifi-framework



(a) Wi-Fi PMF Deauthentication Vulnerability. **(b) All-Zero Key Reinstallation Vulnerability.**

Figure 2: Example console output of the test cases presented in the demonstration of our framework.

Since manual patching is required, or the usage of the development branch, some stations may still be vulnerable to the attack.

Test Case. In order to evaluate if an access point is vulnerable to the deauthentication attack, we define a test case which operates as a supplicant and implements CVE-2019-16275. That is, we must send a plaintext association request to the access point, using a broadcast source MAC address. As the first action, using a successful connection as its trigger, we send a plaintext frame on the monitor interface. If the access point is vulnerable, it replies with a PMF-protected broadcast deauthentication frame, disconnecting all its client stations. As a result, we can define a second action which triggers when our station is disconnected from the network, and if so, we know the attack was successful and the access point is vulnerable. An example of a vulnerable station is given in Figure 2a.

3.2 All-Zero Key Reinstallation Vulnerability

Researches identified a number of key reinstallation attacks [9], commonly known as KRACK attacks, in which an adversary abuses design or implementation flaws in cryptographic protocols. These weaknesses allow an adversary to reinstall an already-in-use key and as a result decrypt network traffic. Depending on the network configuration, the vulnerabilities may even be used to inject and manipulate data. Notably, certain stations can be forced to install an all-zero encryption key, making frames trivial to decipher.

Test Case. In order to evaluate if a client is vulnerable to the installation of an all-zero pairwise encryption key, we define a test case which operates as an authenticator and executes the key-reinstallation attack. That is, we must resend the third message of the four-way handshake. As the first action, using a successful connection as its trigger, we invoke a user-defined function. In this function, we use *wpaspy* to issue a command to the *hostap* control interface, asking the access point to resend the third message of the four-way handshake. Upon receipt of this message, a vulnerable client will install an all-zero pairwise encryption key. We then define an action which processes any frame received by the client under test, and evaluate if a frame is encrypted with an all-zero pairwise encryption key. Upon detection of such a frame, we have verified that a client is vulnerable. An example is given in Figure 2b.

4 CONCLUSION

We presented a framework to swiftly test and fuzz Wi-Fi devices by creating a language in which complex test cases can be defined. Our framework leverages the control interface of the *hostap* user space daemon, and uses and extends its built-in commands. As such, researchers are no longer required to implement complex features or procedures from scratch. Our framework allows researchers to test hypothesis on new weaknesses, implement proof-of-concepts, build fuzz-testing tools, create testing suits, or automate experiments.

REFERENCES

[1] Aldo Cassola, William K Robertson, Engin Kirda, and Guevara Noubir. 2013. A Practical, Targeted, and Stealthy Attack Against WPA Enterprise Authentication. In *NDSS*.
[2] Cas Cremers, Benjamin Kiesl, and Niklas Medinger. 2020. A Formal Analysis of IEEE 802.11's WPA2: Countering the Kracks Caused by Cracking the Counters. In *29th USENIX Security Symposium (USENIX Security 20)*. 1–17.
[3] Jouni Malinen. 2019. *AP mode PMF disconnection protection bypass.* Retrieved 14 Mar 2021 from https://w1.fi/security/2019-7/ap-mode-pmf-disconnection-protection-bypass.txt
[4] Amirali Sanatinia, Sashank Narain, and Guevara Noubir. 2013. Wireless spreading of WiFi APs infections using WPS flaws: An epidemiological and experimental study. In *2013 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 430–437.
[5] Domien Schepers, Aanjhan Ranganathan, and Mathy Vanhoef. 2019. Breaking WPA-TKIP Using Side-Channel Attacks. In *Black Hat Europe Briefings, Location: London, UK*.
[6] Domien Schepers, Aanjhan Ranganathan, and Mathy Vanhoef. 2019. Practical Side-Channel Attacks against WPA-TKIP. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. 415–426.
[7] Chris McMahon Stone, Tom Chothia, and Joeri de Ruiter. 2018. Extending automated protocol state learning for the 802.11 4-way handshake. In *European Symposium on Research in Computer Security*. Springer, 325–345.
[8] Mathy Vanhoef. 2021. Fragment and Forge: Breaking Wi-Fi Through Frame Aggregation and Fragmentation. In *Proceedings of the 30th USENIX Security Symposium*. USENIX Association.
[9] Mathy Vanhoef and Frank Piessens. 2017. Key reinstallation attacks: Forcing nonce reuse in WPA2. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1313–1328.
[10] Mathy Vanhoef and Eyal Ronen. 2020. Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 517–533.
[11] Mathy Vanhoef, Domien Schepers, and Frank Piessens. 2017. Discovering logical vulnerabilities in the Wi-Fi handshake using model-based testing. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*.
[12] Wi-Fi Alliance. 2018. *Wi-Fi Alliance introduces security enhancements.* Retrieved 30 May 2020 from https://www.wi-fi.org/news-events/newsroom/wi-fi-alliance-introduces-security-enhancements