TALLINN UNIVERSITY OF TECHNOLOGY
School of Information Technologies
Department of Software Science

ITC70LT
Alberto Zorrilla Garza 177279IVCM

BEACONLEAK: USE AND DETECTION OF 802.11 BEACON STUFFING AS A COVERT CHANNEL

Master's Thesis

Supervisor: Olaf Maennel

Ph.D. (Dr.rer.nat.)

Tallinn 2019

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutisüsteemide instituut

ITC70LT
Alberto Zorrilla Garza 177279IVCM

BEACONLEAK: 802.11 MAJAKKAADRISSE PEITMISE MEETODI KASUTAMINE JA TUVASTAMINE PEITKANALINA

Magistritöö

Juhendaja: Olaf Maennel

Ph.D. (Dr.rer.nat.)

Tallinn 2019

Author's declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Alberto Zorrilla Garza

2019-05-16

Abstract

The IEEE 802.11 standard defines ”Elements” in the beacon frame specification as sections that are used to relay specific access point information, the most common one being the SSID or ”WiFi network name”. A technique known as "beacon stuffing" allows embedding extra information in the Elements of a beacon frame. This technique has been used before for information broadcasting, advertising, location based services and data exfiltration. In this work we present ”beaconLeak”, an open source tool to demonstrate the feasibility of beacon stuffing as a covert channel for the later stages of the cyber intrusion kill chain. This work also presents the functionality to detect the use of this method, which upon detection, generates log events for security correlation and monitoring tools to consume. Additionally, this thesis presents attack scenarios, risks and threat analysis while also providing measurements and evaluations of the implementation to prove the viability of ”beaconLeak” as both an attack and defense artifact of beacon stuffing as a covert channel.

This thesis is written in English and is 74 pages long, including 5 chapters, 24 figures, and 7 tables.

Annotatsioon

IEEE 802.11 standard defineerib elemente majakkaadri spetsifikatsioonis kui sektsioone, mida kasutatakse edastamaks spetsiifilist pääsupunktide informatsiooni, kus üheks kõige tavalisemaks näiteks on traadita kohtvõrgu (WiFi) ja ta seadmete identifikaator ehk võrgunimi (SSID). Tehniline meetod "Majakkaadrisse peitmine" on meetod, mis võimaldab peita informatsiooni majakkaadri elementidesse. Nimetatud tehnikat on eelnevalt kasutatud informatsiooni leviedastamiseks, reklaamimiseks, asukohapõhiste teenuste tarbeks kui ka andmete välja viimiseks ettevõttest. Lõputöös esitletakse tööriista "beaconLeak", lähtekoodtarkvara demonstreerimaks majakkaadri parameetritesse informatsiooni sokutamise võimalikkust kui potentsiaalselt peitkanalit kübersurma ahela hilisemates etapides.

Lisandväärtusena on töös välja arendatud funktsionaalsus majakkaadrisse peitmise meetodi kasutamise tuvastamiseks, mis lisaks tuvastamisele, genereerib ka vajalikud logid, võimaldamaks luua erinevatele seire-ning turvalahendustele vajalikke korrelatsioone.

Lisaks vaadeldi töös võimalikke rünnakustsenaariume, viidi läbi riski- ja ohuanalüüs, teostati mõõdistused ning anti hinnangud juurutamisele, et oleks võimalik kinnitada "beaconLeak" tööriista elujõulisust majakkaadrisse peitmise meetodina seda nii ründe- kui ka kaitsemeetmena.

Lõputöö on kirjutatud Inglise keeles ning sisalda teksti 74 leheküljel, total 5 peatükki, 24 Figuret, 7 tabelit.

List of abbreviations and terms

AES Advanced Encryption Standard

AP Access Point

APT Advanced Persistent Threat

BSSID Basic Service Set Identifier

C2 Command and Control

CBC Code Block Cipher

CPU Central Processor Unit

CTS Clear to Send

DLP Data Loss Prevention

DRAM Dynamic Random Access Memory

DSRM Design Science Research Methodology

IDS Intrusion Detection System

IoC Indicator of Compromise

IPS Intrusion Prevention System

IT Information Technologies

LIDS Link Layer Intrusion Detection System

LOS Line of Sight

LTE Long Term Evolution

MAC Media Access Control

NIC Network Interface Card

NLOS Non-Line of Sight

OS Operating System

PARADIS Passive Radiometric Device Identification System

PoC Proof of Concept

QoS Quality of Service

SDR Software Defined Radio

SIEM Security Information and Event Management

SSID Service Set Identifier

TIM Traffic Indication Map

USB Universal Serial Bus

USB OTG Universal Serial Bus On-The-Go

VM Virtual Machine

VPN Virtual Private Network

WIDS Wireless Intrusion Detection System

Contents

List of Figures

List of Tables

1 Introduction

Cyber security incidents over the last ten years have signaled the inevitable involvement of state sponsored threat groups in the internet and its systems' security [1]. Armed with better funding and more ambitious goals, state actors have completely innovated in what used to be considered a realm for the technical specialists only. The eventual recognition of the cyber domain as a new domain for strategy and conflict engagements has demonstrated the creation of military units with the sole purpose of covering this new domain in their strategic multi-domain operations [1]. The advent of these advanced threat agents have brought with them a new understanding of these group's capabilities and ingenuity to achieve their objectives [2].

Limitation is an excellent drive for creativity, and when the main objective is to remain undetected, we find the most creative solutions to achieve so. Specifically, we are referring to side or covert channels, which can be used for both commandeering an attack and achieve extraction objectives. These indirect methods have been observed and utilized by the security community in the latest incidents. Just these last year's examples like the family of Spectre [3] and Meltdown [4] vulnerabilities in the Central Processor Unit (CPU) design, or the Row Hammer Dynamic Random Access Memory (DRAM) memory attacks [5]. All these attacks utilized side channels to either extract data or modify the main channel to expose confidential information.

Covert or side channels [6], also called Out of band, present a refreshing insight into the ubiquity of information around us. Further spectacular work, like M. Guri's et. al. "bridging the air gap" series of papers, drive a further point that there is always a side channel to uncover [7]. It would seem that the idea of securely confining information has to be discarded when considering the myriad of ways our own computers constantly leak information, even from their base design, making it very difficult to come up with a truly air tight threat model.

This has inspired us to investigate a network covert channel, and this work stems from the single simple question: Could we use the WiFi network name to extract confidential data?

In this Thesis, we will demonstrate a wireless covert channel attack while also presenting the means for detecting such attack.

1.1 Ethics

The artifact presented in this work can be both used for attack and defense. The purpose of this is to provide Red Teams and Security Operation Centers with the means of setting up detection and testing this detection of the beacon stuffing method. The evaluations presented in this work are the result of running this tool in a laboratory environment without causing any harm to other networks. The capture of information was done with passive capturing and special care was used to only capture wireless beacon frames, which are considered public information as they are broadcast by surrounding networks. To add some perspective, the information capture method is exactly the same to the method a smartphone uses to present surrounding WiFi network names when looking for one to connect.

1.2 Identifying Problem and Motivation

1.2.1 Research Motivation

When designing secure infrastructure it is common to think that the information assets will remain safe in a computer if such computer is never connected to the internet, and in general this is a safe assumption. However in practice, it has been observed that when crafting our threat models, these assumptions have hidden vectors of attack, which still eludes analysis. Advanced threats have uncommon techniques; understanding these techniques will reveal the ineffective approach usual to common threat modeling. This sheds light into the massive endeavor that developing a threat model against state level adversary is. Without considering the fundamental design and architecture of our hardware and software, we will always be blind to the possibilities hidden within them.

The main motivation for this research stems from reading the work done by Mordecai Guri et. al. on their research paper series know as "bridging the air gap", some examples include utilizing computer fans [8], router LEDs [9], hard-drive noise [10], USB electromagnetic emissions [11], hard-drive LEDs [12], thermal deltas [7], computer speakers [13] and infrared cameras [14] as channels for command and control or data exfiltration. On these papers, Guri et. al. show how previously uncovered side channels were capable and usable to extract confidential information from machines without any network connections. We would assume that these techniques have been tried in the wild by advanced actors to complete their information acquisition objectives, however, cheap and ready

made detection methods for these techniques are rare or non-existent, as is the evidence to the usage of these techniques.

Information is all around us, and our devices have the ever expanding capabilities, each with its own flaws and strengths. It is because of this that we turn to WiFi. Its ubiquity and robustness make it a valuable option as it is difficult to find areas in modern cities or buildings without WiFi coverage, a mobile phone or a laptop computer without a Wireless Network Interface Card (NIC).

Can we use the WiFi Service Set Identifier (SSID) to exfiltrate data from a compromised machine?

The SSID is bounded to 32 octets, however, it is an 802.11 Element, which is defined by the IEEE 802.11 standard as 255 subsections of the beacon frame that carry information about the WiFi access point [15]. Some of these Elements are variable length and reserved for future use, we can abuse the standard and fill them up with our custom data. This technique is called Beacon Stuffing [16]. Can we use this to exfiltrate confidential data?

1.2.2 Research Novelty

Researchers have used this technique in the past to add data into the beacon for various purposes [16][17][18][19]. In 2014, Tom Neaves from Trustwave published a blog post where he details how to use this technique, while also providing functionality description and screen captures of a tool he developed called Smuggler [20]. However, he never released any code and no further information exists about it. On personal communication with Mr. Neaves, he stated that his main goal was that of awareness of this method.

During the writing of this research, it came to our attention that a similar attack tool was quietly released by Yuval Nativ on a GitHub repository for his data exfiltration multi-tool "PyExfil" [21]. The main difference is that Nativ's tool is mostly a PoC of the attack, and has no detection mechanism whatsoever. We will further explore the main differences and show a comparison of our implementation to Nativ's in the evaluation section.

While this attack might seem convoluted at first, with the right capabilities and opportunity, we believe it to be both feasible and undetectable. Preliminary security scenario analysis shows that by lowering the threshold of threat agent capabilities, the impact increases meaningfully, specially when data confidentiality is paramount. We have not

found in the literature any comprehensive security analysis of the beacon stuffing method as a covert channel attack. In this work we present ours.

As previous proof of concepts of this covert channel attack has been described, the novelty of this research resides in the detection, development and evaluation of several Indicators of Compromise (IoCs) and mechanisms related to the attack method. No other detection mechanism for the beacon stuffing as a covert channel has been presented before. No detailed security analysis of the beacon stuffing method as a covert channel attack has been presented before.

1.2.3 Research Questions

Can we use the WiFi beacon frame stuffing technique as a covert channel?

With the beacon stuffing technique in mind, and the previous descriptions of its possible uses, we designed and developed a tool expanding on the capabilities originally described by Neaves [20]. This allowed us to further expand from the original question, into the following:

- Can we extract confidential data through this covert channel?
- Is this covert channel detectable?
- In which scenarios could this method be used?
- In which platforms would this attack be feasible?
- Can Rogue Access Point (AP) detectors be a solution against this attack method or covert channel?
- What is the speed of transfer of this covert channel?
- If this is detectable, what would be valid indicators of compromise?
- Will this method work on air-gapped systems?
- What is the practical physical proximity for this technique be usable?

1.3 Scope

This work explores the viability of the beacon stuffing technique as an adversary's attack method in the 2 latest stages of the cyber kill chain [2]. This means that our scope will not deal with the specifics of the initial foothold of the system or actual system infection.

A risk assessment scenario is included in the feasibility study to illustrate the scope of the attack. The proposed solution includes the design and development of the tool that utilizes this technique using a Red Teaming approach.

Also, a viable detection method was designed and developed to provide a usable sensor that generates basic indicators of compromise. These can be processed by security monitoring, detection and correlation tools to support security operations.

A mode to inspect previously captured packets for IoCs was also developed, aiding forensic efforts where the tool has not been deployed or extending capabilities of existing solutions.

1.4 Research Domain

The main domain of this research is cyber security, however, we would like to point out several subdomains which overlap within the work being presented. In 2017, Henry Jiang published a mind map of cyber security domains, with further collaboration, he presented the version 2.0 of his map which in our opinion has a very clear illustration of the subdomains components that form cyber security super-set [22]. Being provided with a clear overview by this map, we consider the research to be rooted in the following sub-sets of the cyber domain.

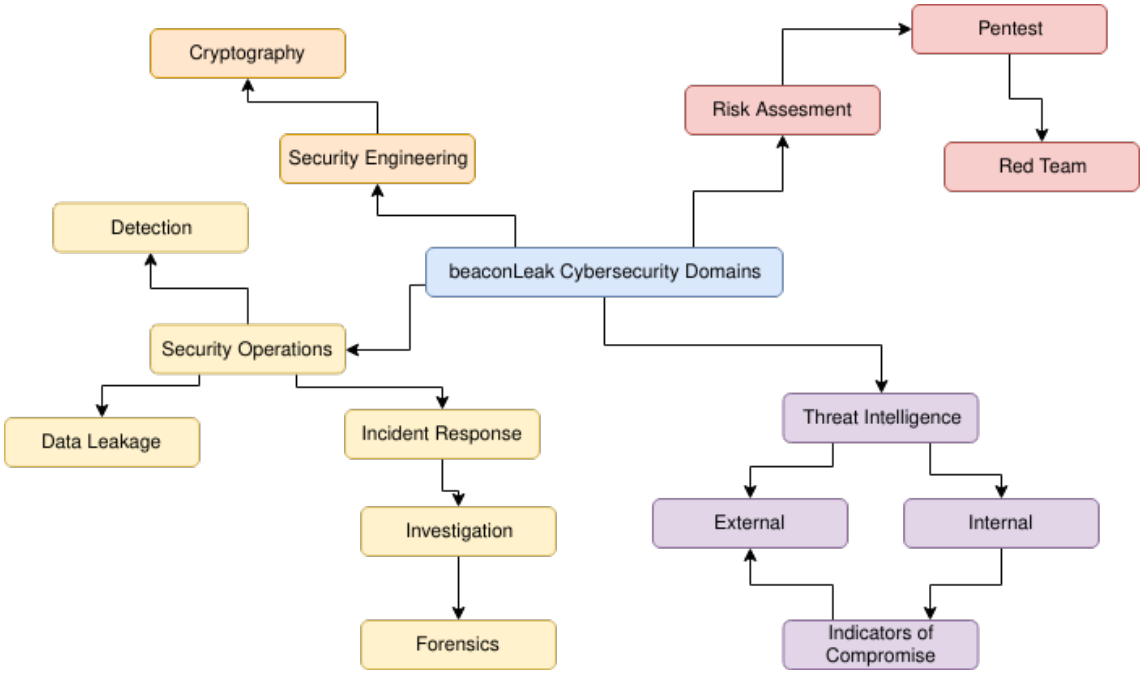

Figure 1. beaconLeak on Jiang's Cyber Security Domain 2.0 [22].

1.4.1 Risk Assessment

The main approach for this research is starting from an adversary's perspective. This technique is known as Red Teaming. We selected this approach as the Red Team main's goal is to act as the attacker and from this mindset develop a prototype which then is tested in the general scope of system analysis. This is best exemplified by the original Air Force Red Team's objective to imagine, develop a prototype and test future threats systems [23].

Criteria The main advantage of taking this perspective is that it allows to simulate the possible actions by the adversary and mitigate against them. This proves to be a very effective way of discovering novel attack techniques.

1.4.2 Threat Intelligence

Once the attack has been imagined and a prototype developed, the next step is to assign an scenario where the adversary might use the new technique as a valid method to secure their objectives. This culminates with the necessary information needed to fully develop

a robust threat model.

Criteria Mavroeidis et. al. described cyber threat intelligence as knowledge about existing or potential threats backed with empirical proof or evidence [24]. This sets a clear addition to the methodology to generate this evidence for the attack previously developed.

1.4.3 Security Operations

With the evidence that an attack is undergoing, the information has to be tied to different areas of security operations. These areas include the operations of Detection, Intrusion detection systems, Intrusion prevention systems, security information and event management, incident response and security operations centers. We find this to be the direct counterpart for a Red Team approach, known as Blue Team.

Criteria The development of the detection mechanism has to be compatible with security operations. The proposed solution should then be useful in this subdomain by creating the atomic indicators of compromise of the developed attack, making the proposed solution a viable sensor for Intrusion Detection System (IDS), Intrusion Prevention System (IPS) and Security Information and Event Management (SIEM) software. By using a ”pcap” file offline analysis, we also support forensic work.

1.5 Methodology

For this work, we are taking a design science approach with the goal of generating a valid artifact. Using the Design Science Research Methodology gives the advantage of having both software tests and experimentation as proper evaluation methods [25].

With this approach, our proposed solution will be able to reach the artifact's development objectives while answering the research questions mentioned earlier. We find the process defined by Peffers et. al. for their Design Science Research Methodology (DSRM) model to be fairly straightforward [25].

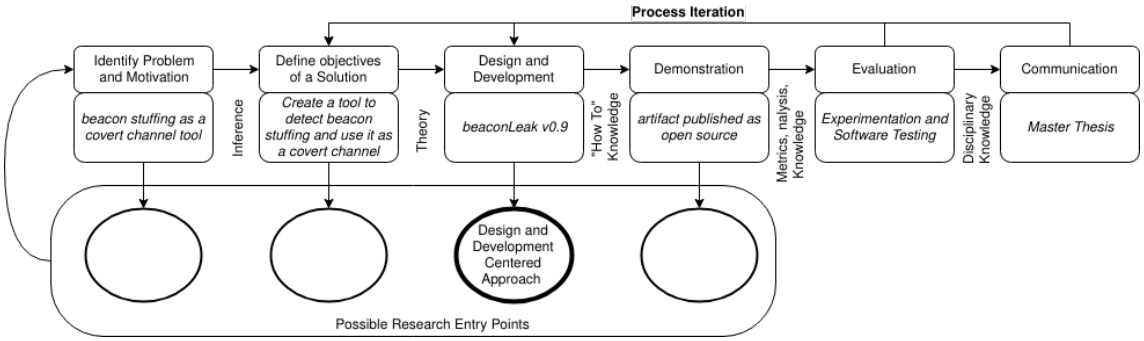

Figure 2. beaconLeak DSRM Process as defined in [25, Figure 1].

This research entry point is the Proof of Concept (PoC) code we first developed to test the beacon stuffing method. This PoC led us to our current research questions.

To illustrate the process in the methodology, the following list defines which parts of the DSRM process are located in relation to this work sections.

- Identify Problem and Motivation: Section 1.2
- Define Objectives of a Solution: Section 1.6
- Design and Development: Chapter 3
- Demonstration: Section 3.7
- Evaluation: Chapter 4
- Communication: Chapter 5

1.6 Objectives

Following the DSRM process [25], the following are the main objectives of this research work:

- Develop a tool that proves the viability of beacon stuffing as an attack method for the later stages of the cyber kill chain.
- Develop the capability into this tool to also detect and generate indicators of compromise for security correlation and monitoring tools to consume.
- Evaluate different aspects of the implementation to further prove the viability of beaconLeak as both an attack and defense solution.

2 Background

In this section, following the DSRM process [25], the theory in the literature is explored and analyzed in an iterative manner to improve the design and development of the design artifact.

2.1 State of the Art

The IEEE 802.11 standard defines the beacon frame as a management frame for broadcasting wireless access points, the beacon frame contains the necessary information to begin association and authentication to the wireless network, this information is contained in a structure called Element [15].

Ranveer Chandra et. al. presented in their 2007 research an introduction to the "beacon stuffing" technique, describing it as a method to communicate without network association by embedding data into the beacon frame [16]. To do so, they proposed embedding the data with three different techniques: SSID Concatenation, Basic Service Set Identifier (BSSID) Concatenation and Beacon Information Element [16]. Further analysis by Gupta et. al. revealed the SSID concatenation to be undesirable in the context of information bandwidth and proposed utilizing the newest defined vendor specific elements [17].

Sven Zhel et. al. then provided a practical implementation of beacon stuffing for location based services, presenting a working prototype of a beacon based bulletin board which utilizes the stuffing technique to broadcast relevant information directly from router software [18].

In 2011, Gonçalves publishes his master's thesis for the Monterey California Naval Postgraduate School on using the 802.11 Media Access Control (MAC) Frame as a covert channel, including a working python implementation as proof of concept [26]. While similar in nature, Gonçalves' work focuses on a different 802.11 frame, specifically the Clear to Send (CTS) frame [26]. Furthermore, the covert channel described by Gonçalves is meant to work through the network.

In 2012, Sawicki and Piotrowski identified possible usage of Elements in the beacon frame for a covert channel, but the application they proposed was for authentication purposes, noting the channel was one-way only [27].

In 2014, Tom Neaves described in Trustwave's SpiderLab Blog his tool "Smuggler", an example on using the beacon stuffing technique for the purpose of covert communication and data exfiltration [20]. Neaves is the first to propose utilizing the beacon stuffing technique as a two way covert channel [20], he presented example code to build a beacon frame with the Scapy Python program and provided some screen captures of the tool at work as a command and control shell [20]. The proposed tool was never released publicly, but the researcher did made a note on the shortcomings of his own design. in personal communication with Mr. Neaves, he declares that the main goal for presenting this method was awareness.

Smuggler creates a two-way covert channel by using the beacon as a command and control medium, used to send execution commands, and having the victim send expected output as response in plain text through the probe frame [20]. The researcher also describes on his publication a tool called Anti-Smuggler, which is able to detect the information transmission through the usage of string search and regular expressions in the probe frames [20]. It is important to note that both tools were only described yet never released.

In March 2018, Yuval Nativ added a commit to his PyExfil tool repository. He described the functionality as a wireless method for physical data exfiltration [21]. The main functionality includes an Advanced Encryption Standard (AES) 128 bit encryption in Code Block Cipher (CBC) mode (AES128-CBC), for the data being sent using the 802.11 beacon frame Traffic Indication Map (TIM) element, his covert channel is one-way only [21]. This work was found after the design and implementation of our own proposed solution for attack, predating ours by at least a year. None of Nativ's work was used in the design of our artifact, but the tool was considered in our detection mode. A tool comparison is provided in the evaluation section of this work.

Latest analysis of the beacon stuffing technique for non-security related operations has been demonstrated by Reijba et. al. in their "Fog-to-Cloud" systems deployment analysis paper [19].

2.2 Threat Intelligence

Threat intelligence deals with the detection and classification of threats by identifying key aspects of threat operations with evidence-based knowledge [24]. Using the Modified Maturity Level Model proposed by Mavroeidis et.al. [24], we can illustrate that in the

context of Threat Intelligence, the following are important to understand about a threat agent:

- Identity
- Goals
- Strategy
- Tactics
- Techniques
- Procedures
- Tools
- Host and Network Artifacts
- Atomic Indicators

In the context of this work, we are delivering the possible Identity, Goal and Technique of the threat agent we simulate for creating the artifact. Furthermore, the artifact provides the Tool and Atomic Indicators this threat agent might use. This all derived from taking a Red Teaming approach for the creation of our artifact [23].

2.2.1 Advanced Persistent Threats

In 2013 Mandiant, now FireEye, published an extensive analysis on the actions of the People's Republic of China Military Unit 61398, assigning them the acronym "APT1" for Advanced Persistent Threat (APT) [1]. In their work, the authors conclude their investigation affirming that this Chinese military unit employs hundreds of personnel with the specific mission of gaining unauthorized access through the network to steal confidential information from different industries, focusing on those in The United States of America [1]. FireEye makes a point that they're considered "persistent" as the authors observed that they resided in a victim's network up to 4 years [1].

Since then, more than 40 different APTs have been identified by their signature use of malware, techniques or technologies. Most of them are considered either state level actors or operating with nation state support [28].

2.2.2 Cyber Security Intrusion Kill Chain

First proposed by E. Hutchins et. al. from the Lockheed Martin Corporation [2], the kill chain was presented as a way to understand the process of an intrusion event, giving detail of the different stages of intrusion that advanced persistent threats take when looking for objectives on a system. The kill chain is composed by the following seven stages:

1. Reconnaissance
2. Weaponization
3. Delivery
4. Exploitation
5. Installation
6. Command and Control
7. Actions on Objective

Objectives are specific to each campaign and each victim, but the general idea is that these threat actors are looking for data or systems to take into their control.

2.2.3 Objectives

In the context of a threat agent, the motivation and goals are the drivers for the strategy [24]. This is important as the complexity of the attack, the risk of detection and the objectives will have an effect on the threat agent's attack method. In the context of this work, a very specific set of objectives must be met for an attacker to use the method described.

2.3 Air Gap Systems

In his article *"International conflict and cyberspace superiority: Theory and practice"*, Bryant proposes the use of air gapping as a technique for critical systems' defense [29, p. 107]. The main idea is that a computer that is completely disconnected from the internet is much more difficult to attack. In the context of threat modelling and information security assurance, this is an effective control against the most basic types of threat actors,

however, more advanced threats have the capability to jump this air gap to complete their objectives.

2.3.1 Relevant Real Life Scenarios

The most recurring example of an advanced threat agent targeting an air gapped system comes from the dissection of the malware known as ”Stuxnet”. In their article ”Lessons from Stuxnet”, Chen et. al. present the conclusion that the level of sophistication and complexity of the tool pointed at a nation state level threat actor [30]. Furthermore, the malware was so complex and the objective so specific that it is now considered a cyber weapon [31]. Stuxnet beat the air gap in their target machines by being executed from a USB drive [30].

This documented event gives credence to the idea that active threats are looking for ways to innovate on the cyber domain to achieve their objectives. In the context of this work, we place ourselves in the position of the threat to present a technique to achieve these kind of objectives.

2.3.2 Bridging the Air Gap

This work is in part inspired by the air gap attack papers presented by Mordechai Guri et. al. from the Ben-Gurion University of the Negev. In these papers, the researchers show different ways to leak data in a covert manner using side channels previously not considered. A few examples include utilizing computer fans [8], router LEDs [9], hard-drive noise [10], USB electromagnetic emissions [11], hard-drive LEDs [12], thermal deltas [7], computer speakers [13] and infrared cameras [14], as channels for command and control or data exfiltration.

In the context of this work, we present beacon stuffing as a technique to jump the air gap, given that the limitations of presence of a wireless adapter and physical proximity are met. Like much of the work presented by Guri et. al., physical proximity is needed. The main similarity of the beacon stuffing technique to this work is the usage of covert channels to carry out data exfiltration and avoid detection. The main difference with the work of Guri et. al. is that we are utilizing a covert channel in a network layer.

2.4 Covert and Side Channels

The notions of covert channels is certainly not new. Kemmerer defined a covert channel as transmission mediums on entities or objects that were not originally intended to be used as communication channels [32]. In his analysis of the CPU side channel attacks, Vateva-Gurova et. al. make a clear point that the term "side channel" and "covert channel" can be used interchangeably [6]. Furthermore, Veteva-Gurova et. al. also make the key observation that covert channels are difficult to detect because they are usually not in the scope of traditional detection systems [6].

In the context of this thesis, we present beacon stuffing as a covert or side channel exactly for these key reasons. The beacon is not intended as a two-way communication channel nor expected as such, by definition it is an information broadcast frame. Also, no current detector exists to analyze a beacon data for hidden information.

2.5 Data Exfiltration

In their work about data exfiltration and covert channels, Giani et. al. observe that the set of possible exfiltration methods would need to include, at a minimum, the set of all communication methods [33]. As a result, Giani et. al. propose a generalized taxonomy for data exfiltration methods as a framework to organize and develop mitigation against them [33, Figure 3]. Using their proposed taxonomy, Figure 3 illustrates how the designed artifact would be categorized.

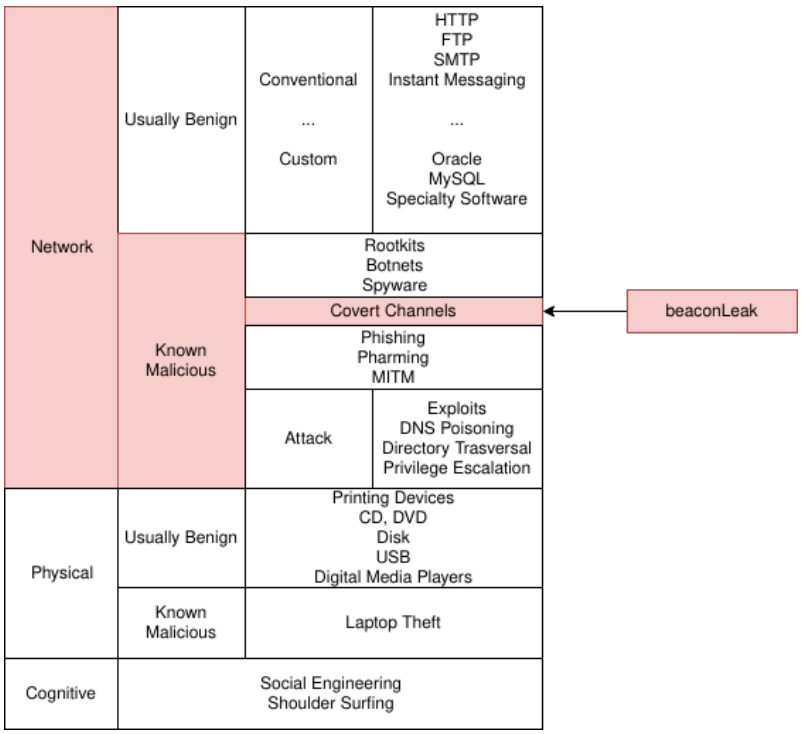

Figure 3. beaconLeak in the Data Exfiltration Method Taxonomy as defined in [33, Figure 3].

2.6 Intrusion Detection and Prevention Systems

Software known as IDS has the main task to identify attempts by an intruder to compromise security goals of a system or to detect activity that is not common on a system [34]. According to an aspect of IDS classification proposed by Sobh, they can be identified by their system type: Network based, Host based or Hybrid [34]. Even though lower layer IDS have been described, they are uncommon [35]. A specialized form of LIDS is the Wireless Intrusion Detection System (WIDS), which has a special monitoring focus for attacks on elements of the 802.11 specification [36]. WIDS focus specifically on beacon frame attacks, however, even the newest proposed systems do not account for covert channel attacks on the beacon frame [37].

The main difference between an IDS and an IPS is that the preventive system will also have the capability to act on the detected intrusion by means of blocking access, dropping packets or other reactive action, while detection systems usually only raise alarm upon detection [34].

In the context of this work, beaconLeak operates in a lower layer of the network, and a

layer specific to wireless networks, making it invisible to detection systems focused on network packets. By the categorization proposed by Zaman et. al. [35], beaconLeak detection mode is both a WIDS and a Link Layer Intrusion Detection System (LIDS).

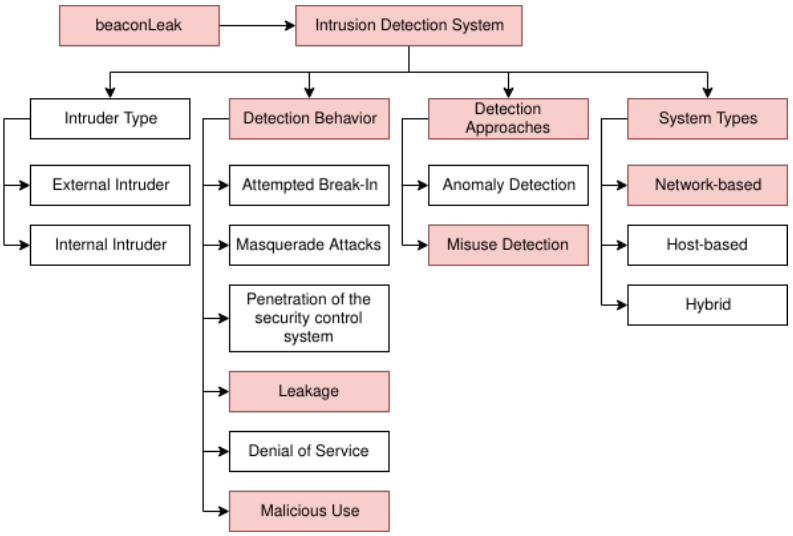

Figure 4. beaconLeak in the Classification of Intrusion Detection Systems as defined in [34, Figure 3].

2.7 Data Leakage Prevention

As mentioned before, threat agent's objectives usually include the acquisition of confidential information. As a reaction to this, security software vendors have responded with a new category of security tool known as Data Loss Prevention (DLP). Hauer noted in her excellent article that vendors use the term DLP to advertise products which are vague about their specific capabilities [38]. In this same work, Hauer addresses the categorization of DLP in their focus to the state of data [38, p. 2555].

- Data at Rest
- Data in Use
- Data in Motion

Hauer also points out that these solutions often struggle with the automatic classification of data into these categories, and that DLP has to use techniques such as: key words and regular expressions detection, digital fingerprints, data tagging and machine learning [38, p. 2556]. Furthermore, on the topic of channels, and specifically covert channels, Hauer

observes that identifying them is a major challenge due to their complexity and covert nature [38, p. 2560]. Another observation made in the study is that covert channels are a constant threat to DLP solutions [38, p. 2563], as the covertness of the channel can be increased in relation to the amount of data sent through it, directly referencing the same study we have used to locate beaconLeak in the data exfiltration taxonomy [33].

In the context of this work, we can assume that a DLP solution monitoring data at rest would be able to detect access to the data, but would fail to detect the covert channel and thus the exfiltration of the information using the designed artifact because wireless beacons are out from DLP software scope [38, p. 2562].

2.8 Rogue Access Point Detection

Rogue APs Detectors are specialized WIDS that make a detailed analysis of wireless beacons. This inspection is done because a Rogue AP main mode of attack consists on adding a wireless access point to a wired network. Another specific attack, known as ”evil twin attack” or ”fake AP attack” will mimic an already existing AP to make a client connect to the malicious AP instead. In their work on fake access point detection, Kao et. al. demonstrate a technique that takes into account the interval, serial number, and timestamp of sniffed beacons frames to determine if a fake AP exists [39].

Kao et.al. proposes Rogue AP Detectors to be classified into two categories [39]:

- Packet Analysis
- Radio Frequency Sniffing

Rogue AP detectors are the detection mechanisms that more closely resemble a detecting solution for the stuffed beacon covert channel, specifically if these detectors have any sort of correlation between the elements of a transmitted beacon, as the one presented by Kao et. al. [39]. Another practical example, sentrygun, is a Rogue AP detection software with the capability to detect evil twin attacks by identifying anomalies in signal strength [40]. However, more simple detectors like Gonçalves’ [41] or similar, which focus on specific 802.11 attacks and MAC Address whitelisting, are easily defeated [39].

In the context of this work, a special "covert" mode was developed to bypass MAC Address whitelisting Rogue AP detectors. It has to be noted however, that more specialized

detectors like the one proposed by Kao et. al. [39] and those that measure physical signal properties like the one proposed by Brik et. al. [42], could detect the beacon stuffing covert channel. Our rationale is that these detectors are still deficient as they would detect the covert channel as either a ”Fake AP” or a false positive due to the beacon frame timestamp.

To completely avoid more advanced types of Rogue AP detectors, the covert channel can be done in the Probe Request frame instead.

3 Design and Development

Following the DSRM process [25], the design artifact is designed and developed in an iterative manner.

Why is beacon stuffing a covert channel? How is it that beaconLeak can use this method as a covert channel and also detect it? The main point, as illustrated in Figure 5, is that you can hide information on a layer protocol data unit that is outside of the visibility of common security monitoring and correlation tools.

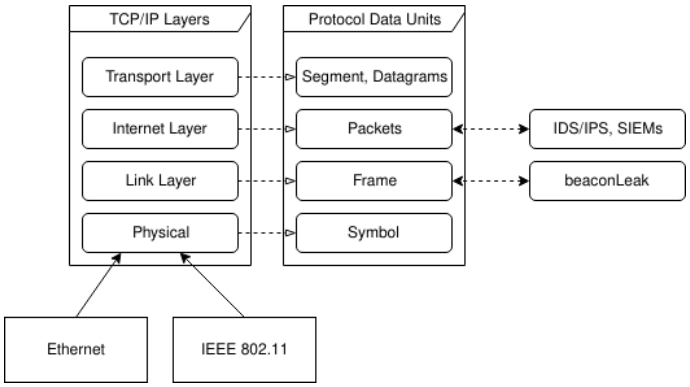

Figure 5. beaconLeak in the TCP/IP Model.

3.1 Threat Analysis

Considering the cyber attack Intrusion Kill Chain, first proposed by E. Hutchins et. al. from the Lockheed Martin Corporation [2], this method of covert communication would mainly be used by attackers in both the ”Command and Control” phase as well as the ”Actions on Objectives” phase [2].

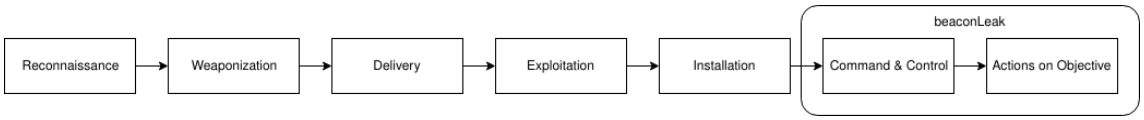

Figure 6. beaconLeak in the Lockheed Martin’s Cyber Kill Chain [2].

It is supposed that the network environment is heavily monitored. Another assumption is that the attacker would have the possibility to reach the wireless signal, meaning that physical proximity is needed. This method would better suit an insider threat, however, external threats could also benefit from it.

3.1.1 Security Risk Analysis

We start by doing a risk analysis to fully understand the position of beacon stuffing as an attack method, beaconLeak as a tool providing this method, for a threat agent looking to steal sensitive data. For this analysis we are using the risk analysis method proposed by Matuleviçius [43, p. 31]. This method starts by identifying the business assets, system assets and the security criteria for these assets [43, p. 33]. The next step is to analyze the threat, attack method, vulnerability and threat agent to finally identify the impact to the business assets created by our threat [43, p. 33-39].

The purpose of this analysis is to illustrate a general threat scenario where the beacon stuffing covert channel is used as the attack method, identify potential threat agents and the possible impact of their actions when the main business asset is data.

Table 1. Risk Analysis - Asset Definition and Security Criteria.

Business Asset	System Asset	Security Criteria for Business Asset
Sensitive Data	Device which has 802.11 wireless capabilities	Confidentiality of the sensitive data

Table 2. Risk Analysis - Insider Threat.

Security Risk Analysis		
	Selected Risk Component	Instantiated Risk Component
System Asset	Data Management	Device which has 802.11 wireless capabilities
Vulnerability	Environment	Layer 2 out-of-band communication is out of scope from data leak prevention systems and network monitors
Threat Agent	Class: Insider. Expertise: Specialist. Capability: Infiltration, Physical Access, Wireless Proximity. Motive: Business or National Interest. Opportunity: Internal access to sensitive data and infection of a wireless capable device on a monitored environment.	An insider who infected a wireless capable device with hardware access privilege and sensitive data access stuffs information in the 802.11 beacon frame. This data is then covertly broadcasted on the beacon frames allowing for the insider to exfiltrate data from a physical wireless proximity undetected.
Threat and Attack Method	Threat: Uncontrolled Operation Attack Method: "Beacon Stuffing" Data Exfiltration	An insider utilizes beaconleak on a device with 802.11 wireless capability. The malicious software uses the WiFi card to inject beacon frames. Using beacon stuffing, sensitive data is hidden in 802.11 beacon frames. The insider can then remotely, from a wireless distance, gather the data without triggering security monitor alerts.
Impact	Data Leak	Negates internal sensitive data confidentiality.

Security Risk Definition - Internal Threat An insider threat agent with specialist expertise capable of malicious software development, infiltration and physical proximity infects a device which has 802.11 wireless capabilities and access to sensitive data. Using the ”beacon stuffing” method the attacker is able to evade data leak prevention software and network security monitors as they are unable to detect layer 2 ”out-of-band” communications. This event leads to a leak of sensitive data, negating information confidentiality.

Observations The insider threat is the most insidious as knowledge of internal processes makes the attack more feasible to pull off. The insider would also have an advantage taking into account that an insider would probably wouldn’t need to mind the first stages of the cyber security kill chain, as the insider itself is already an infiltrated agent, the kill chain would only start from the fifth step of "Installation" [2].

Table 3. Risk Analysis - External Threat.

Security Risk Analysis		
	Selected Risk Component	Instantiated Risk Component
System Asset	Data Management	Device which has 802.11 wireless capabilities
Vulnerability	Environment	Layer 2 out-of-band communication is out of scope from data leak prevention systems and network monitors
Threat Agent	Class: External Attacker. Expertise: Specialist, Advanced. Capability: Malware development, Infiltration, Physical Proximity, Wireless Signal Proximity. Motive: Business or National Interest. Opportunity: Foothold on a system with access to sensitive data and infection of a wireless capable device on a monitored environment.	An external attacker who infected a wireless capable device with hardware access privilege and sensitive data access stuffs information in the 802.11 beacon frame. This data is then covertly broadcast on the 802.11 beacon frames allowing for the insider to exfiltrate data from a physical wireless proximity undetected by security monitoring software.
Threat and Attack Method	Threat: Uncontrolled Operation Attack Method: "Beacon Stuffing" Data Exfiltration	An attacker utilizes a malware plant on a device with 802.11 wireless capability. The malware plant uses the WiFi card to inject beacon frames. Through beacon stuffing, the frames are utilized for data exfiltration. The external attacker can then remotely from a wireless distance gather data without triggering monitors alerts.
Impact	Data Leak	Negates internal sensitive data confidentiality.

Security Risk Definition - External Threat An external threat agent with specialist expertise capable of malicious software development, infiltration and physical proximity

infects a device which has 802.11 wireless capabilities and access to sensitive data. Using the ”beacon stuffing” method the external attacker is able to evade data leak prevention software and network security monitors as they are unable to detect layer 2 ”out-of-band” communications. This event leads to a leak of sensitive data, negating information confidentiality.

3.1.2 Attack Scenarios

Internal A cyber operations agent ”Eve” is tasked with leaking a specific file that lives on a laptop in the adversary state’s bank Information Technologies (IT) development office. This file is a private key used by developers to sign their software packages and releases. The intelligence report given to ”Eve” states that the laptop is air-gapped but has a disabled wireless NIC. An insider succeeded to infect the machine and gain an initial foothold on it, installing beaconLeak to the target which is now waiting for instructions. Further intelligence tells ”Eve” that the adversary state’s bank is running technical interviews for developers; the interview room is in wireless proximity to the laptop. ”Eve” sets beaconLeak on an android phone which is then hidden in one of the candidate’s bag. beaconLeak in the phone issues a file download on the target machine, acquiring the laptop’s file without triggering network monitors. ”Eve” connects remotely to the phone through the Long Term Evolution (LTE) network and is then able to tunnel the file to cyber command. The phone is remotely wiped.

External An adversary state diplomat is visiting the capital. It is known by cyber command that the ensign carries sensitive data in his laptop, however, any detection would raise tensions and create a diplomatic nightmare. The previous operation succeeded in installing malware on the diplomat’s computer with an ”evil maid” attack, but the operative revealed that the target system is currently connected through the hotel’s Ethernet to the adversary’s state diplomatic Virtual Private Network (VPN) and any movement or control through the net would alert the endpoint DLP and the remote network monitoring systems. Luckily cyber command deployed beaconLeak and left it waiting for instructions. From the floor above, the cyber operative is able to stealthily extract all sensitive files using beaconLeak masquerading all beacon frames as the hotel’s WiFi. Upon getting what was needed, the cyber operative commands beaconLeak to destroy the delivery malware and itself from memory without leaving any trace.

3.2 Functionality

We developed a command line tool with both attack and detection capabilities, intended to be used by red teams and blue teams in their tool set. The following is a description of the functionality implemented.

3.2.1 Command and Control (C2) Mode

This is the mode that an attacker would use to control target devices. In Command and Control (C2) mode, an attacker can control multiple victims or single machines in an intended range. The way beacons work allows to broadcast messages so multiple machines can receive and interpret these as OS commands. This is a stateless covert communication channel that allows to send executable commands to the operating system in the victims machine. Optional flags allow to either expect a response or simply send a command without waiting for the output reply (one-way or two-way channel). Furthermore, this mode contains a special command to download files from the victim's device.

3.2.2 Leak Mode

Leak mode allows to simulate an infected victim, it runs an infinite listener that will continuously scan beacons for a stuffed beacon. When a stuffed beacon is found, the mode will attempt to decode the communication. When a valid communication packet is found, i.e. the decryption of the covert message is possible, this mode will act on the command received. This mode can execute commands received, send the output of commands using the same stuffing technique and send files. Without any extra flags, leak mode is completely silent, producing no output of any kind to best emulate an infected victim.

3.2.3 Detection

Detection mode works in two modes: offline, for analysis on packet capture files, or online, for live 2.4GHz WiFi beacons auditing. Several "indicators of compromise" have been included to detect different beacon stuffing methods. When a stuffed beacon is detected, this mode will raise a log event to "Syslog" in Linux systems.

3.2.4 Debug Flag

For extra verbosity, all modes accept a debug flag that will print extra information. This is useful when the selected mode is not working as intended an extra information is needed to understand when or where in the process is the program failing. On leak mode, it activates output that otherwise is hidden.

3.3 Detection Indicators of Compromise

For the detection mode to be useful, it should have a very specific definition of the ”indicators of compromise” to avoid false positives and false negatives. The following is a list of the IoCs implemented into detection mode and a rationale of why they were chosen. This also sets the test bed for detection’s evaluation.

3.3.1 Beacon Size

If every vendor is following the 802.11 specification, it would make sense that there is an average size to a beacon frame, and deviation from this size could be treated as an anomaly.

Rationale It is logical that a beacon stuffed with extra information will have a large size delta compared to those not stuffed. This size threshold should prove to be a very efficient way to detect extra information hiding on the beacon. Without a detection mechanism, there is no visibility about the specifics of a WiFi beacon other than that implemented in it original use of AP station technical specifics. When analyzing a beacon size, and comparing it to the expected values, finding anomalies in size can be considered as an indicator of compromise for this attack.

3.3.2 Reserved Elements

Any use of reserved Elements should be treated as an IoC. It is described in the 802.11 specification that vendors should use Element ID 221 to add any extra data not specified on the standard.

Rationale The beacon stuffing technique, as mentioned, works by using elements of the beacon frame to stuff extra information for communication or exfiltration. While this can be done on specified elements such as the SSID or Rates element, non-standard or unexpected information would make packet sniffers like Wireshark to flag the frame as ”malformed”. Because of this, reserved elements, which are unused, present a very attractive container for our information as they are simply shown as ”unknown tagged”, as shown on Figure 12.

3.3.3 Tool Default Configuration

As with most software that has configuration options, there is a notion of configuration defaults. In the case of some attack tools, these default configuration options might provide a way to detect the usage of a specific tool, and certainly the specific configuration of that tool.

There are only two known available implementations of this attack method, the one presented in this work and PyExfil [21], then one described but never released [20]. These implementations, including our own, use a default configuration unless the threat agent specifies otherwise. Those defaults could be treated as an IoC for detection.

Rationale Most low lever attackers will use a tool without reading the specification or manual, expecting to carry their attack with only a couple of example commands. From a usability perspective, it makes sense to have a tool that is easy to use, tending to more complex use cases only when needed. These defaults are an excellent indication of the tool being used.

3.4 Design

3.4.1 Design Decisions

There are three main quality considerations for the development of this software as a usable tool:

- **Portability**: The code should be able to work regardless of the underlying operating system. the goal is having the tool work in at least Linux and Windows

environment.

- **Security**: as T. Neaves pointed out in his publication, he was able to detect his tool only by searching for strings [20]; to avoid this, the communication between victim and attacker should be confidential, using modern strong cryptography.
- **Functionality**: The tool should be capable to both use the method as an attacker would and detect the attack happening in its surroundings, it should provide a mechanism to generate a simple indicator of compromise using the system log, to be consumed by security monitoring or correlation tools.

3.5 Software Architecture

The tool was developed using an Object Oriented Programming approach, allowing for future developers to extend the tool or integrate the tool as a module for larger attack or defense frameworks. This decision was made to keep secondary design goals of modularity and extensibility.

3.5.1 Diagrams

In Figure 7, we present a simplified version of the beaconLeak Python class and its dependencies.

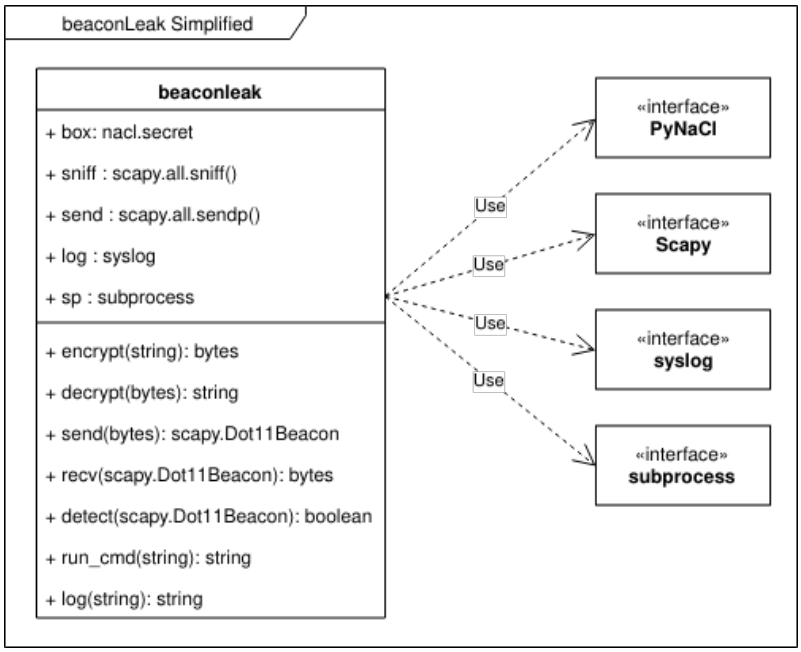

Figure 7. beaconLeak simplified Class Diagram.

The main use cases and actors of the program are visualized on the use case diagram in Figure 8.

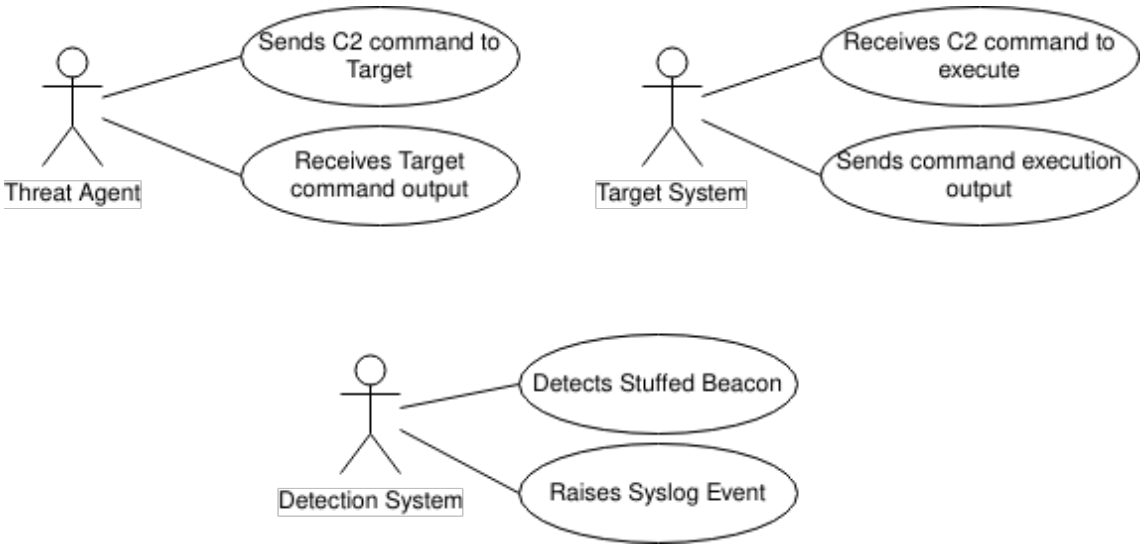

Figure 8. beaconLeak main Use Cases Diagram.

In the deployment diagram represented in Figure 9, we can visualize the involved nodes and the data flow between them.

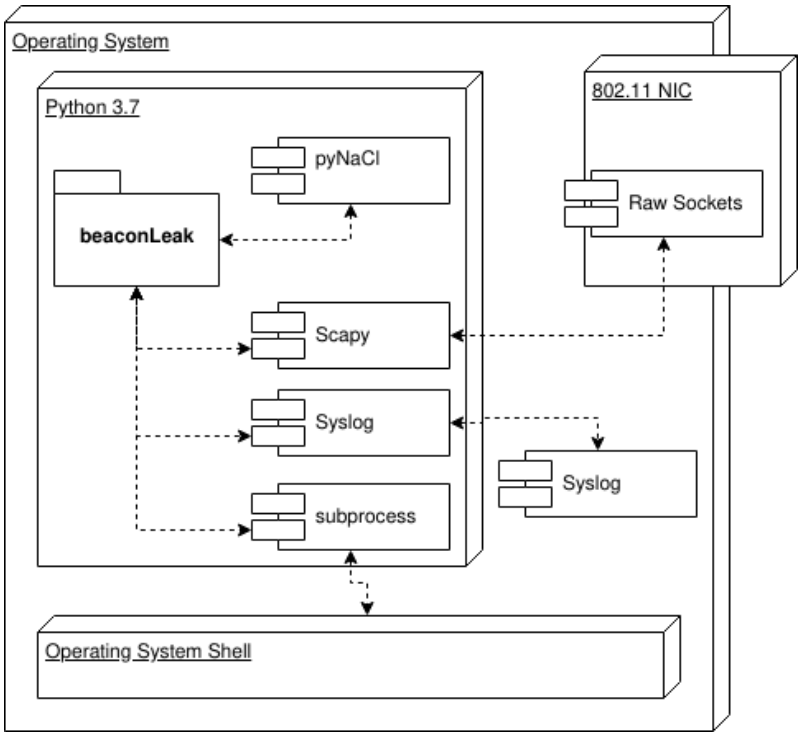

Figure 9. beaconLeak Deployment Diagram.

3.5.2 Data Transmission

A covert channel was built on the physical layer of the network, but this doesn't mean that the covert channel has a network stack. This means that we only have a medium, and that we need to create our own solutions to problems that usually the network stack takes care of, like Quality of Service or data transmission integrity. Our implementation is not the exception, we have a channel on a wireless medium, but we need to develop a solution to ensure data transmission. To show an example of how we have solved communication issues in a new channel without the network stack, the communication diagram of the use case "Threat agent downloads file from Target System" is visualized in Figure 10.

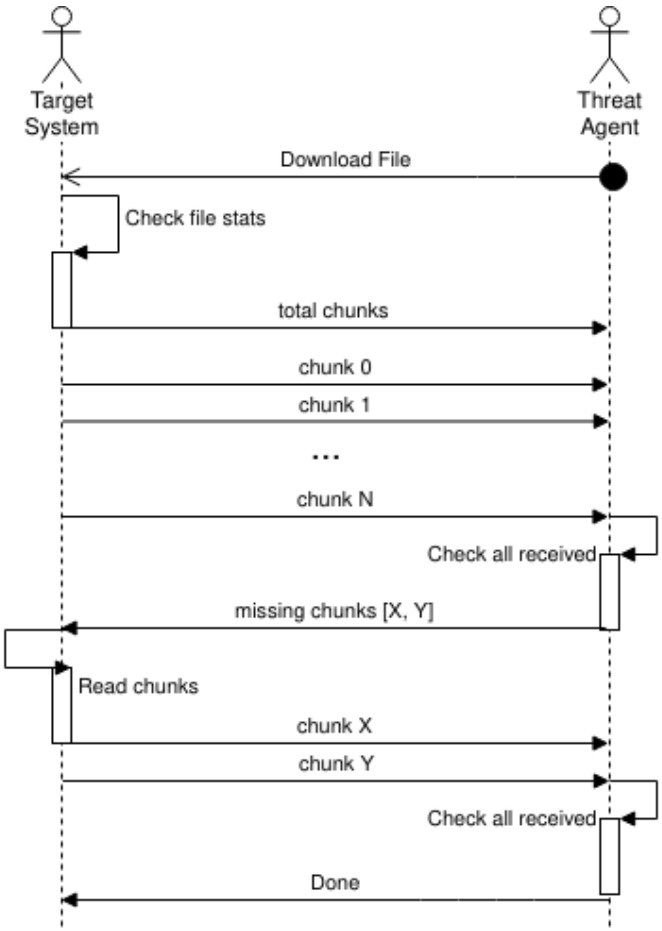

Figure 10. Download use case communication diagram.

The process illustrated in Figure 10 shows the step-by-step download procedure implemented in our attack. Threat agent is the actor using the tool in "C2 Mode" while Target System is the actor running our tool in "leak mode". When a download command is received by the leak mode, the file is measured and split into chunks, these chunks are then sent to the tool running in "C2 mode", to be sequentially decoded and built back into the original file. As shown in the figure, the most basic Quality of Service (QoS) is used: re-transmission of lost chunks.

3.6 Technical Aspects

3.6.1 Programming Language

The Python language version 3 provides a good balance to develop programs in a rapid manner with the support of mature libraries while providing the necessary design considerations sought. Its richness in multiple libraries provides an excellent environment to quickly develop proof of concept code.

3.6.2 Network Packet Forging Library

Scapy is a Python program with network packet transmission and reception capabilities [44]. The beaconLeak tool utilizes Scapy as a library, for it provides the capacity of forging network packets, satisfying functionality for the main use case of beaconLeak.

It is this program that allows the tool to create a stuffed beacon frame and transmit it through the wireless card. Part of the functionality that we added is the covert mode, where beaconLeak can listen to surrounding beacons and use the strongest one to stuff it and hide from simple rogue access point detectors.

3.6.3 Cryptographic Library

After a thorough search for cryptography libraries, we decided to settle with PyNaCl [45], an interface for libsodium [46], which is the implementation of the NaCl cryptography library [47]. It was the clear choice as it was developed to provide strong confidentiality, modern cryptography, fast operation speeds and usage simplicity [48].

This library was developed with the specific goal of avoiding the cryptography pitfalls of other libraries, like OpenSSL, as demonstrated by Daniel J. Bernstein, et. al. in their introductory paper [49].

Another important reason why this library was chosen, is that its high level abstraction is designed so that the possibility of developer error in the use of this cryptography library is minimal, as shown by Yasemin Acar, et. al. in their study of cryptography libraries usability [50]. For simplicity, we decided to use Pre-Shared key, symmetric cryptography mode, future work should including adding a timestamp to avoid message replay attacks

[51]. The underlying symmetric cryptography algorithm is used in authenticated mode, utilizing the Salsa20 stream cipher and the Poly1305 message authentication code [48, Secret Key Encryption].

3.6.4 Beacon size calculation for detection avoidance

An interesting issue we came across was that, once the frame size IoC was developed and tested, we had to change our transmission method to always account for the beacon frame size to stay ”under the radar”. This seemingly simple issue had an underlying complexity, specially for the calculation of the free space left in the frame to transmit our payload for download stuffed frames.

The logic is the following: we have certain amount of overhead for the creation of a valid stuffed beacon, this leaves us with a fixed space to fit our data. If data is encrypted, we have to also account for the cryptography overhead. The hidden complexity lies in the fact that we also have to tag the frame for sequences of the chunked data in order to have a basic QoS re-transmission of lost frames, so that data extraction is actually useful. The problem is that a dynamic amount of data will yield a variable amount of tags, and thus, the relationship between free space and file size is recursively affected. e.g. calculating the number of tags for the data affects the amount of free size, so a recalculation has to be done ad-infinitum.

To solve this issue, and after some analysis, we came up with the following simple algorithm:

1. Calculate an approximation of the free size left in the frame.
2. Calculate the amount of space needed for our encoded and encrypted tags.
3. Re-calculate the approximation with this new magic number.
4. Calculate the chunks needed with the calculated size.

The algorithm can be represented by the equation in Equation 3.6.4.

$$\lambda = y - O_f - O_{c1} - O_{c2} - O_{l1} - O_{l2}$$
$$C_{total} = \frac{x}{\lambda - (\log_{10}(\frac{x}{\lambda}) + 1)}$$

Where λ = approximation, x = file size in octets, y = max beacon size, O_f = frame overhead, O_{c1} = First cipher overhead, O_{c2} = Second cipher overhead, O_{l1} = First frame layer overhead, O_{l2} = Second frame layer overhead, C_{total} = total chunks needed for file size x.

Which ended up translating into the function in Figure 11

```
def calculate_chunks(file_size, beacon_frame):
  max_size = 325
  approx = max_size - len(beacon_frame) - 84
  magic = math.ceil(math.log10(file_size / approx) + 1)
  r_size = approx - magic
  chunks = math.ceil(file_size / r_size)
  return chunks
```

Figure 11. Function to calculate chunk size for stuffed beacon frames.

3.7 Implementation Demonstration

Following the DSRM process [25], this is the demonstration phase. The code has been published under an open source license and can be found in the following link: `https://github.com/cjcase/beaconleak`

3.8 Limitations

These are the key limitations for this technique to work, regardless of the platform the attack is run from:

- Administrator privileges
- Wireless NIC with monitor mode
- Packet injection capable driver

4 Evaluation

Following the DSRM process [25], the designed artifact is evaluated in an iterative manner. Observations made in this section directly affect design and development decisions on the artifact implementation in an iterative manner.

4.1 Ethics

The capture of information was done with passive capturing and special care was used to only capture wireless beacon frames, which are considered public information as they are broadcast by surrounding networks. To add some perspective, the information capture method is exactly the same to the method a smartphone uses to present surrounding WiFi network names when looking for one to connect, i.e. scanning the 802.11 spectrum for WiFi beacons.

4.2 Implementation Comparison

As mentioned before, the attack has been described previously by Tom Neaves in his post about Smuggler [20], and a PoC implementation by Yuval Nativ for his python exfiltration tool collection PyExfil [21]. These implementations are compared to ours. Notice how our tool is the only one providing a detection mechanism for all implementations, noting that Smuggler was never released [20].

Table 4. Implementation Comparisons.

Implementation Comparison Table							
Implementation	**802.11 Frame Element**	**Encrypted Channel**	**Encryption**	**Detection Mode**	**Detection Mechanism**	**Remote Execution Mode**	**File Transfer Mode**
beaconLeak	Reserved	Yes	Salsa20 and Poly1305	Live, Offline	Multiple IoCs	Yes	Yes
Smuggler	Rates and SSID	No	None	Live	String Regular Expression on element payload	Yes	No
PyExfil	TIM	Yes	AES-256-CBC	No	None	No	Yes

4.3 Cryptography Tests

For simplicity, this test was modeled to both prove the communication between an "attacker" and a "victim" in an environment devoid of a network but where a beacon monitoring sensor exists and evidence that the communication is confidential through the use of cryptography. Evidence of this encrypted communication can be found on Appendix 2 –, lines 21 to 27. This cipher and the key for decryption are included in Appendix 4 –.

- Hostname: eve, Role: Attacker
- Hostname: alice, Role: Victim
- Hostname: bob, Role: monitor

Attacker — Eve has infected Alice, and to test the covert channel a command is issued to gather information about Alice's platform. Please refer to the appendix for an output specimen of the tool.

Victim — Alice computer is in an air gap, but has a WiFi NIC and has been infected with beaconLeak. The malware receives and responds to the command sent by Eve without generating any output as the malware is hidden.

Monitor — Bob is checking the network for unknown WiFi access points, he spots a couple of beacons using tagged elements but the information in the tags seems like gibberish. Bob takes no action. Figure 12 shows Wireshark on bob's station capturing the beacon and showing the contents of the tagged elements.

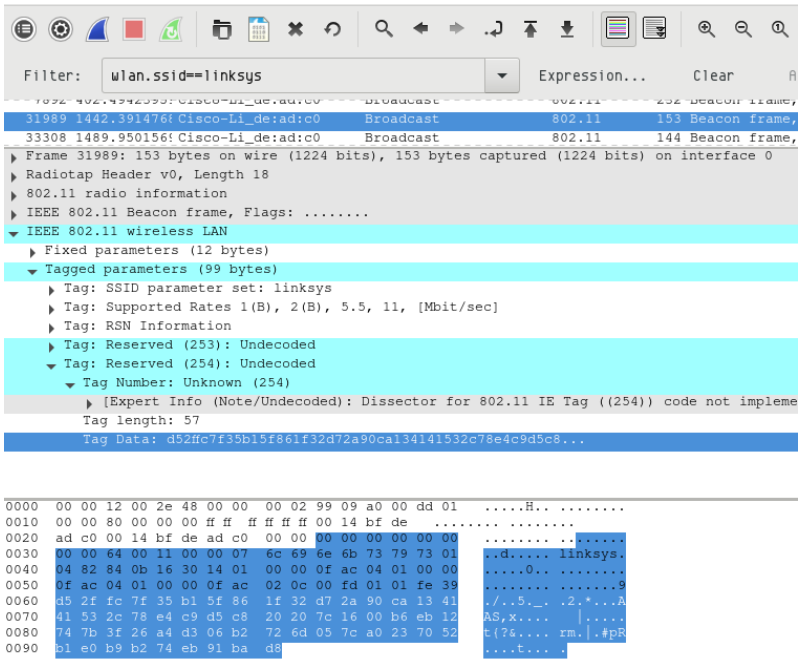

Figure 12. Wireshark capture of 802.11 stuffed beacons.

Further tests were made with a non-default encryption key, please refer to Appendix 4 – for this evidence.

4.4 Portability Tests

For increasing the attack viability, it should work on different platforms. the following is a condensed view for the batch of static tests run on different platforms.

Table 5. beaconLeak Portability Static Analysis.

OS	Leak Mode	C2 Mode	Detect Mode	Packet Injection
Linux	Full	Full	Full	Capable
Android	Partial	Partial	Full	Capable
Windows	Partial	Incapable	Full	Incapable
Mac OS X	Partial	Partial	Full	Capable

The information on Table 5 is a summary of the results of the following evaluations.

Leak, C2 and Detect mode columns refer to the operation modes of the software. Packet Injection refers to the capability of the Operating System (OS) to support packet injection. Three color coded levels are offered: Full, Partial and Incapable in respective green, yellow and red. Partial means that specific hardware or changes have to be done to the underlying operating system in order for the mode to work; Incapable means that there is no know configuration that would allow the OS to use this mode. Full means that no extra hardware or changes to the operating system are needed.

4.4.1 Linux

Test Description This test is a static test that will prove that the technology decision was correct and that the software quality attribute of portability is reached. The program will be run from an Ubuntu 18.04 LTS x64 Desktop Virtual Machine (VM).

Test Process The virtual machine was installed in an Arch Linux x64 host, then, inside the virtualized system, the needed libraries were installed along with the python interpreter. An external USB WiFi NIC was connected to the VM using pass through so it appears natively to the virtualized environment.
The following is the hardware used for this experiment:

- (Ubuntu VM) TP-Link TL-WN7200ND
  Model: TL-WN7200ND(ES) Ver 1.0
  Driver: 802.11n USB Wireless LAN Card (MediaTek 5.1.22.0)
  Manufacturer: Ralink Technology, Corp.
  Factor: External USB 2.0
- (Arch Linux Host) Intel(R) Wireless-N 7265
  Model: Wireless-N 7265
  Driver: Intel Corporation Wireless 7265 (rev 59)
  Manufacturer: Intel Corporation
  Factor: Internal PCI

Further hardware technical specification can be found in Reference [52] and [53].

Result The program is run successfully, the software is able to push the stuffed beacon frames and received by an external device.

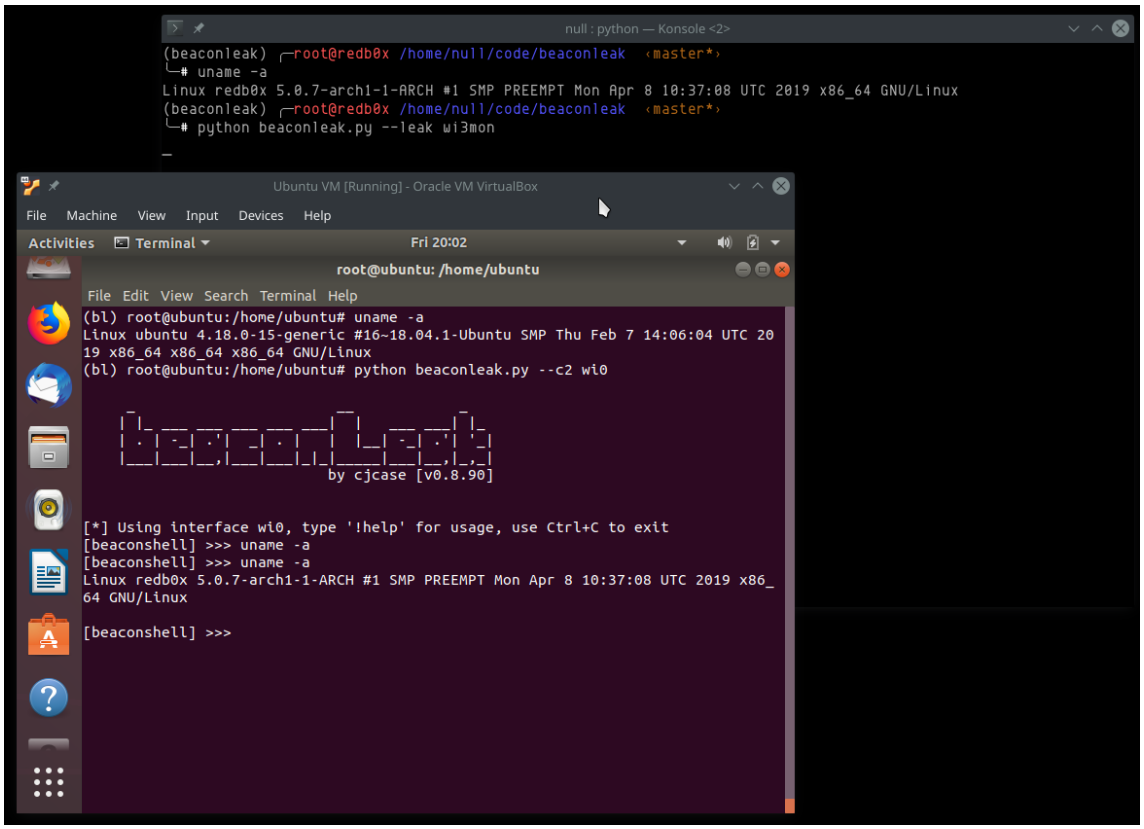

Figure 13. beaconLeak running on Virtual Machine - Ubuntu 18.04.2 LTS.

Evidence Please refer to Figure 13 for a screen capture of the tool working on the OS.

4.4.2 Android

Test Description A mobile device is an excellent candidate as it can create a bridge for extracted data using the LTE mobile network. Also, due to its small form factor, it can double as a cheap stealthy plant to control remote infected machines while its battery lasts.

Test Process For this test, two phones were tested:

- Samsung Galaxy S5 (klte) rooted running Android 8.1.0

- Google Pixel (sailfish) rooted running Android 9

The mobile needs a rooted version of Android for the software to work correctly, then, using the application Termux, we can then install the necessary library dependencies and run the software.

Result The software runs and is able to communicate, however, due to kernel limitations in the phones, an external Universal Serial Bus On-The-Go (USB OTG) adapter was needed to connect a wireless NIC that can be operated in monitor mode with injection capabilities. Because of this, we consider the functionality of our tool on this OS to be partial.

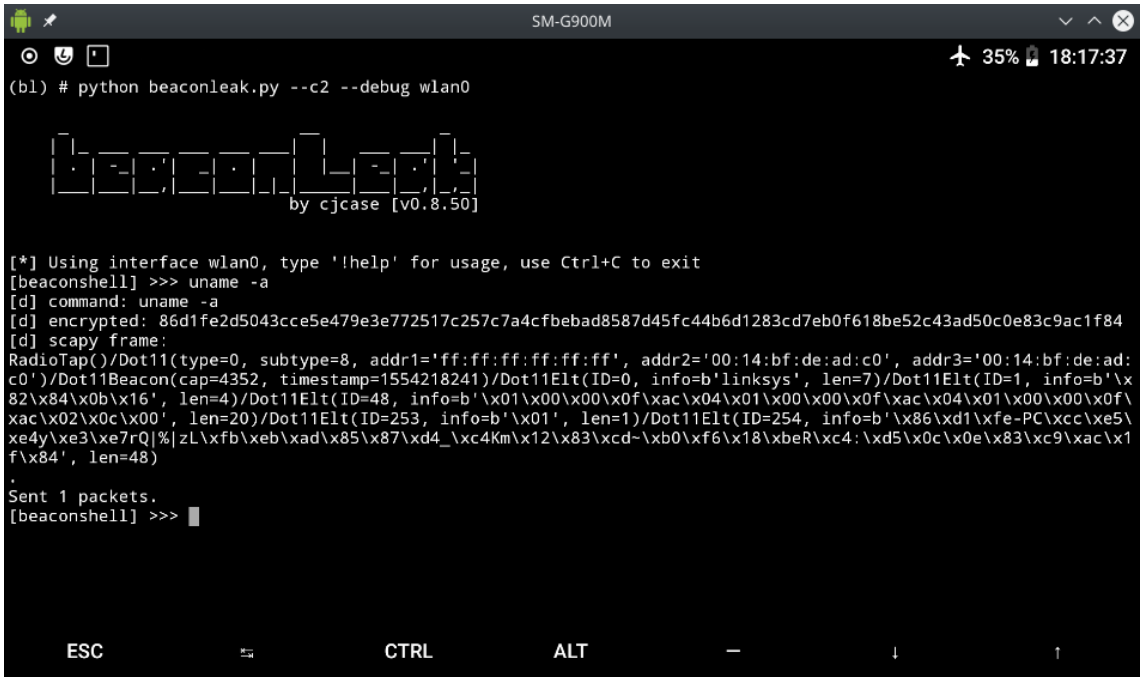

Figure 14. beaconLeak running on Google Pixel - Android 9.

Evidence Please refer to Figure 14 for a screen capture of the tool working on the OS, and to Appendix 3 – for the usage manual developed for this device type.

4.4.3 Windows

Test Description Usually target devices are running different operating systems. Microsoft's Windows is one of the most common OS for consumers. It makes sense to have the software able to run for it increases the feasibility of this technique being run in the wild.

Test Process The software was installed in a laptop running Windows 10.0.14393. Two different Wireless NICs were utilized, one of them an external Universal Serial Bus (USB) NIC.

- TP-Link TL-WN7200ND
  Model: TL-WN7200ND(ES) Ver 1.0
  Driver: 802.11n USB Wireless LAN Card (MediaTek 5.1.22.0)
  Manufacturer: Ralink Technology, Corp.
  Factor: External USB 2.0
- Intel(R) Centrino(R) Advanced-N 6205
  Model: Advanced-N 6205
  Driver: Intel (15.16.0.2)
  Manufacturer: Intel Corporation
  Factor: Internal PCI

Further hardware technical specification can be found in Reference [52] and [53].

Result With the mentioned hardware, no packet injection was capable, rendering the exfiltration aspect impossible. Further research must be done to determine if packet injection is viable in a Windows environment at all. Monitoring mode works on both cards using Npcap version 0.99-r9 bundled with Wireshark.

Please refer to Appendix 3 – for the installation and usage manual developed for this platform.

Evidence Please refer to Figure 15 for a screen capture of the tool working on the OS, and to Appendix 3 – for a manual on how to install and use beaconLeak on this platform.

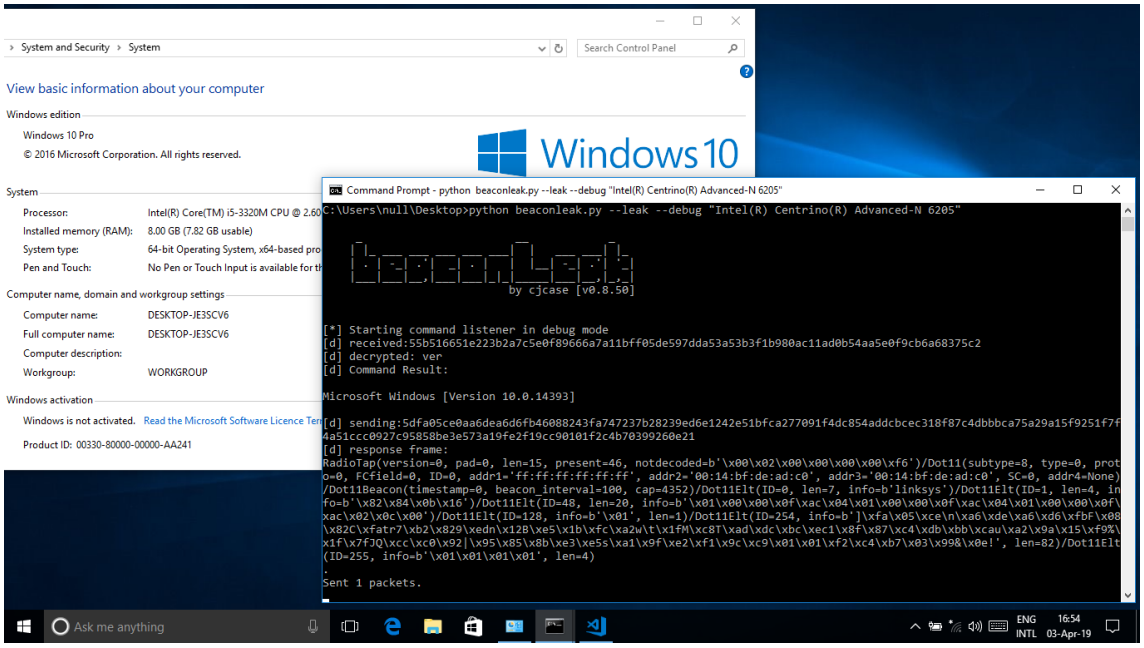

Figure 15. beaconLeak running on Windows 10.

4.5 Detection and Indicators of Compromise

Previously we mentioned the indicators of compromise that we believed were effective for detection in our design of the tool. We presented in the design phase the rationale of each one. The following is the evaluation of these IoCs as detection mechanisms, and in overall of the detection functionality of the designed artifact.

```
# this is for the blue teamers
# IoC detection from simple test to complex test
def detect(self, frame):
  if frame.haslayer(Dot11Beacon):
    ssid = frame.info.decode('utf-8')
    bssid = frame.addr2
    channel = int(ord(frame[Dot11Elt:3].info))
    elements = frame.getlayer(Dot11Elt)
    # IoC: pyExfil defaults
    if frame.addr2 == "00:00:00:00:00:42" or ssid == "
      pyExfil":
```

```
11          msg = f"[!] Beacon Stuffing Detected! (SSID:{ssid}
                BSSID:{bssid} CH:{channel}) [PyExfil defaults]"
12          self.mon_log(msg)
13          return
14        # IoC: beaconLeak defaults
15        if frame.addr2 == "00:14:bf:de:ad:c0":
16          msg = f"[!] Beacon Stuffing Detected! (SSID:{ssid}
                BSSID:{bssid} CH:{channel}) [beaconLeak defaults]"
17          self.mon_log(msg)
18          return
19        # IoC: Beacon size sample std. dev.
20        l_tresh = 174 # these numbers were made with
21        h_tresh = 352 # SCIENCE!
22        l_frame = len(frame)
23        if l_frame > h_tresh:
24          msg = f"[!] Beacon Stuffing Detected! (SSID:{ssid}
                BSSID:{bssid} CH:{channel}) [Beacon Size Treshold]
                "
25          self.mon_log(msg)
26          return
27        # IoC: Reserved Elements
28        while elements:
29          if elements.ID in self.reserved:
30            msg = f"[!] Beacon Stuffing Detected! (SSID:{ssid}
                  BSSID:{bssid} CH:{channel}) [Reserved Element {
                  elements.ID}]"
31            self.mon_log(msg)
32          elements = elements.payload.getlayer(Dot11Elt)
```

Figure 16. Excerpt: Detection function implementing all developed indicators of compromise.

4.5.1 Beacon Size

Hypothesis Can use the size of the beacon as evidence for the usage of the beacon stuffing technique?

Experiment Setup We developed a python script to automatically survey the main 13 channels of the 2.4Ghz WiFi spectrum, spending a minute in each channel and measuring specifics of the beacons in that frequency. Please refer to Appendix 1 – for a copy of the script.

The main goal for this was to use a scientific approach with data support for our size range IoC decision. The script is programmed to save WiFi beacons information using passive sniffing as the acquisition technique, data presented has no identifying information.

Experiment Process We ran this script on three different locations, the captures are available upon request to the author. Summary data is anonymous and contains no personal or sensitive information. Data is presented here and in the software repository:

- A Hotel building.
- Inside of a bank's offices.
- Downtown residential department.

The Hardware used was the same on every location, this is the hardware details specification:

- TP-Link TL-WN7200ND 150Mbps High Power Wireless USB Adapter
- Model: TL-WN7200ND(ES) Ver 1.0
- FCC ID: TE7WN7200ND
- Driver: ID 148f:3070 Ralink Technology, Corp. RT2870/RT3070 Wireless Adapter

Further hardware technical specification can be found in Reference [52]. The python script takes about 780 seconds (approximately 13 minutes) to complete measurements. While running the experiment, efforts were made to avoid interaction with the antennas, their position or moving within the surroundings.

Experiment Results A total of 95543 beacons were captured, which came from 283 unique access points. The mean size of the beacons is 262.67 octets, with a standard devi-

ation of 29.55. This gives us a **lower threshold size of 174.03** and the **highest threshold of 351.31**.

Table 6. Beacon Size IoC Experiment Summary Information.

Total Unique APs	Average Beacon Size @ Hotel	Average Beacon Size @ Bank	Average Beacon Size @ Residence	Median	Mode	Std. Dev.	Curve Lower	Curve Highest	Total Samples
283	264.94	271.05	255.78	264.30	252.00	29.55	174.03	351.31	95543

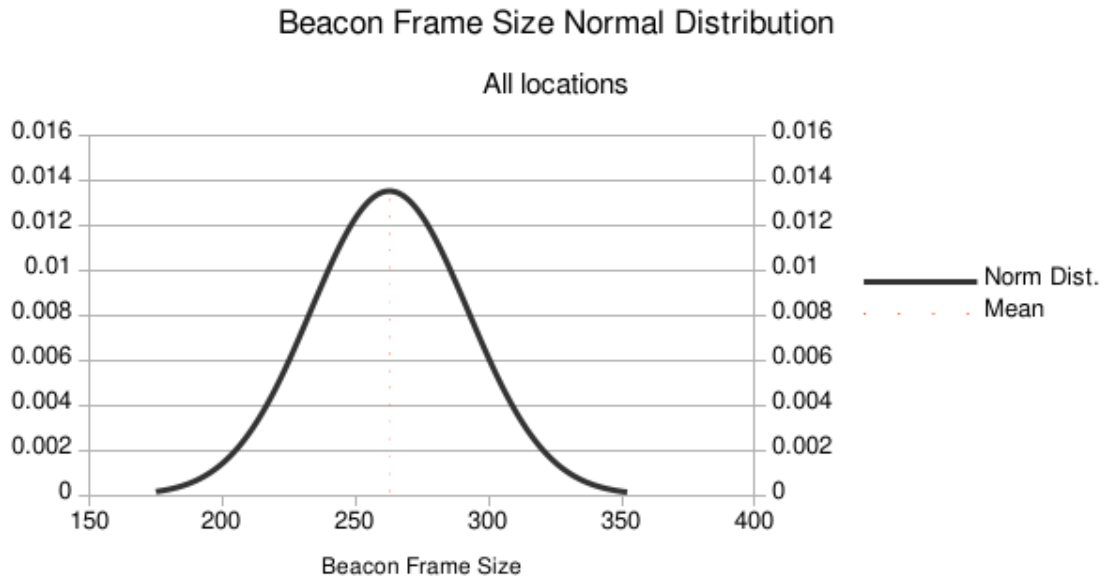

Figure 17. All location's beacon size normal distribution.

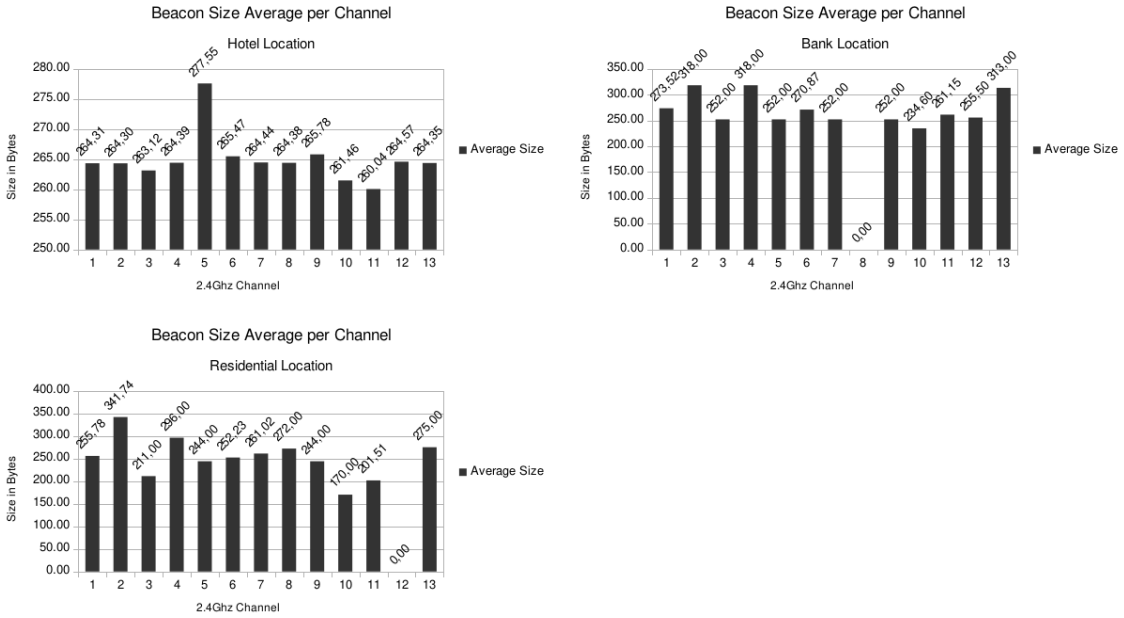

Figure 18. Average beacon size per channel histograms.

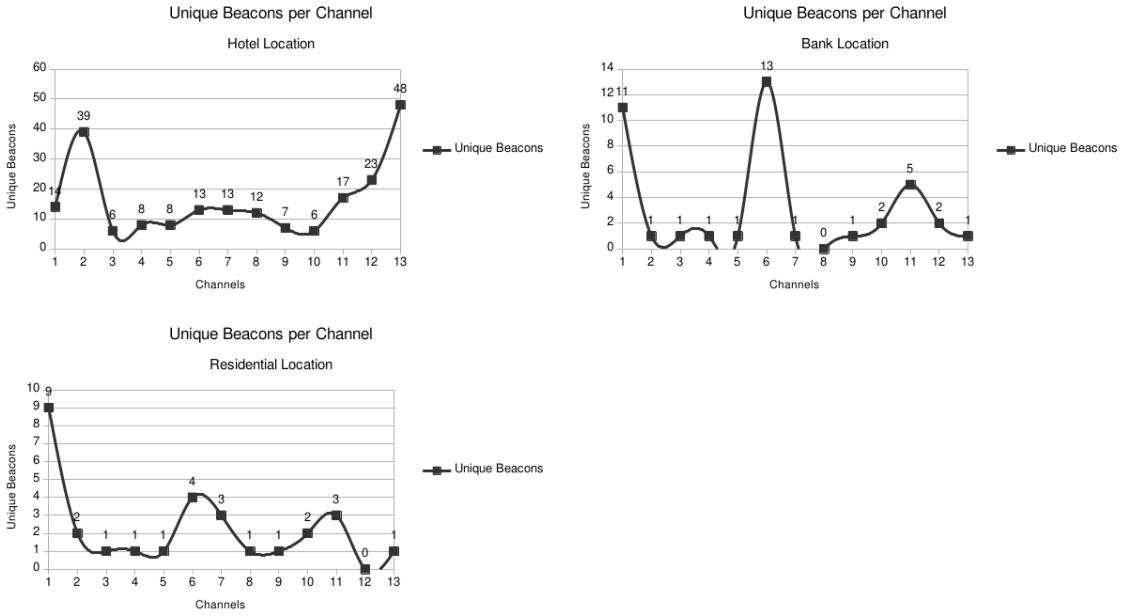

Figure 19. Unique beacons per channel on all locations.

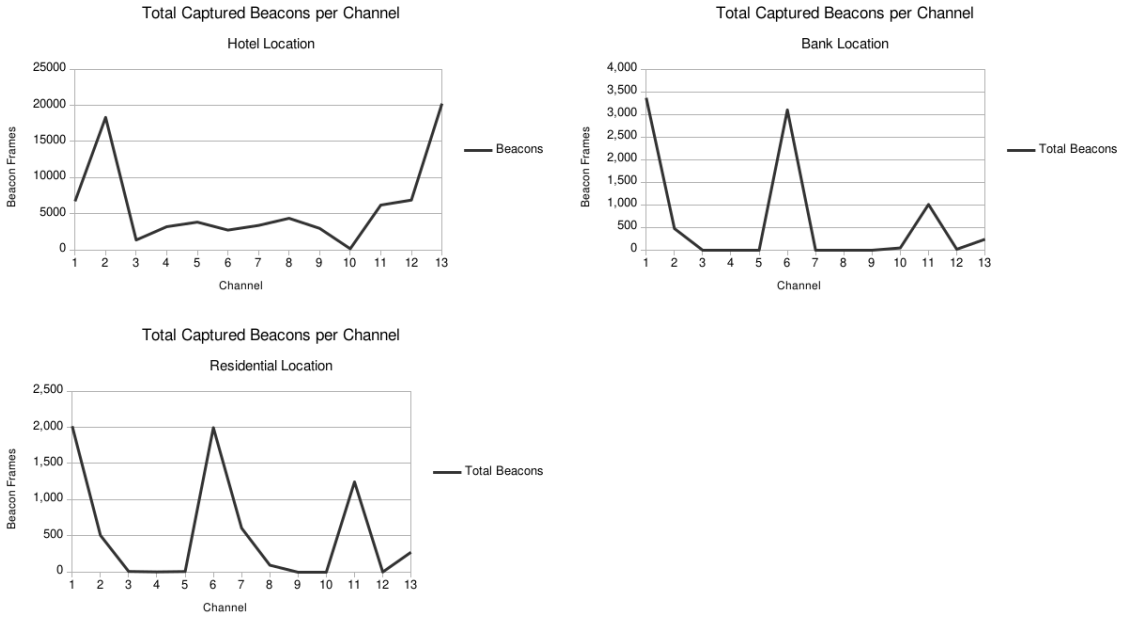

Figure 20. Total beacons per channel on all locations.

Conclusion By introducing a size threshold detection, we are effectively limiting the size of beacon and thus the data transmission bandwidth of the beacons as a communication channel. This limitation is reflected directly on data transfer rates. The presence of such size validations make this method undesirable to attackers seeking higher transfer speeds, which was originally considered an advantage to this method.

The first design of the download functionality concerned itself with data transfer speed. This evaluation proves that **beacons over 351 octets** can be treated as stuffed beacons.

As a direct result of these measurements, and following the DSRM [25], the file download functionality was redesigned to transmit beacons on the defined threshold to avoid detection, with a measurable impact on data transfer rates.

By these observations, size of the beacon frame is a valid indicator of compromise. This has been integrated in the implementation of the proposed solution, please refer to Figure 16 for the implementation.

4.5.2 Reserved Elements

Hypothesis Can the use of reserved elements be used as a detection of the beacon stuffing method?

Experiment Setup We developed a detection mechanism in our artifact. With this we surveyed WiFi beacons to detect the use of reserved elements ”in the wild”. We also developed the beacon stuffing mechanism using reserved elements to test for true positives.

Experiment Process We ran our tool on several different locations, for at least 10 minutes in each location. No packets were saved, merely analyzed in real time for the use of reserved elements.

The Hardware used was the same on every location, this is the hardware details specification:

- TP-Link TL-WN7200ND 150Mbps High Power Wireless USB Adapter
- Model: TL-WN7200ND(ES) Ver 1.0
- FCC ID: TE7WN7200ND
- Driver: ID 148f:3070 Ralink Technology, Corp. RT2870/RT3070 Wireless Adapter

Further hardware technical specification can be found in Reference [52].

Experiment Results During our observations, no other vendors are using reserved elements as specified in the 802.11 standard [15]. Only when we used the beacon stuffing functionality, the reserved elements use was detected correctly.

Conclusion We can treat the usage of reserved elements as a valid indicator of compromise.

4.5.3 Tool Default Configurations

Hypothesis Is there specific default configuration on PyExfil and our solution that we could use as indicator of compromise?

Experiment Setup We did a static code analysis of the PyExfil [21] tool to understand which information was being sent and its mechanism. The default configuration in our tool and in PyExfil transmits specific data on the beacon, i.e. the SSID and BSSID elements, finding them on a beacon we can assume these tools are being used. With this observations and the knowledge of our own design we developed a detection mechanism in our artifact that will find this information on the beacons to flag them as detected. Please refer to Figure 16 for the implementation.

Experiment Process We ran our tool on several different locations, for at least 10 minutes in each location. No packets were saved, merely analyzed in real time for the use of the tools default options.

The Hardware used was the same on every location, this is the hardware details specification:

- TP-Link TL-WN7200ND 150Mbps High Power Wireless USB Adapter
- Model: TL-WN7200ND(ES) Ver 1.0
- FCC ID: TE7WN7200ND
- Driver: ID 148f:3070 Ralink Technology, Corp. RT2870/RT3070 Wireless Adapter

Further hardware technical specification can be found in Reference [52].

Experiment Results During our observations, the usage of both PyExfil and beaconLeak was correctly detected using the IoC hypothesized, while no false positives were detected.

Conclusion We can treat the usage of tool default configurations as a valid indicator of compromise, please refer to Figure 16 for the implementation.

4.6 Proximity and Data Transfer Rate

In our scenarios, we consider the attacker to carry their extraction from either outside close to the victim's premises or still in physical proximity to the infected machine. We tested the transfer at different distances to prove the feasibility of presented scenarios and valid transfer distances.

4.6.1 Experiment

Hypothesis The covert channel can be used to transfer data from a different distances. The main obstacle to the channel will be signal noise.

Experiment Setup Two machines were utilized for this experiment, both running Arch Linux x64 with Linux Kernel 5.0.7-arch1-1-ARCH. The hardware related to radios is the following:

- **Receiver:** TP-Link TL-WN7200ND
  Model: TL-WN7200ND(ES) Ver 1.0
  Driver: ID 148f:3070 Ralink Technology, Corp. RT2870/RT3070 Wireless Adapter
  Manufacturer: Ralink Technology, Corp.
  Factor: External USB 2.0
  Antenna Type: Omni Directional (RP-SMA)
  Antenna Gain: 5dBi
- **Sender:** Intel(R) Centrino(R) Advanced-N 6205
  Model: Advanced-N 6205
  Driver: Intel (15.16.0.2)
  Manufacturer: Intel Corporation
  Factor: Internal PCI

The "Receiver" radio output is rated at 500 mW, further technical specification can be found in Reference [52]. Extra technical specifications for the "Sender" radio can be found in Reference [53].

A file filled with random data was created, with the following specifics:

```
$ dd if=/dev/urandom of=distanceTestFile bs=7K count=3

$ ls -l distanceTestFile
-rw-r--r-- 1 null null 21504 Apr 16 00:31 distanceTestFile

$ sha256sum distanceTestFile
387fa36ab2819232c0de8660465af827c0c1bad767b3478461c55f70693
   64f78  distanceTestFile
```

Then, measurements were made from the different distances as illustrated in Figure 21. The locations were selected as they could be easily measured while also reflecting feasible distances at a Line of Sight (LOS) measurement. For this test, we omitted any re-transmission functionality to properly understand the behavior of the wireless medium in relation to beacon reception.

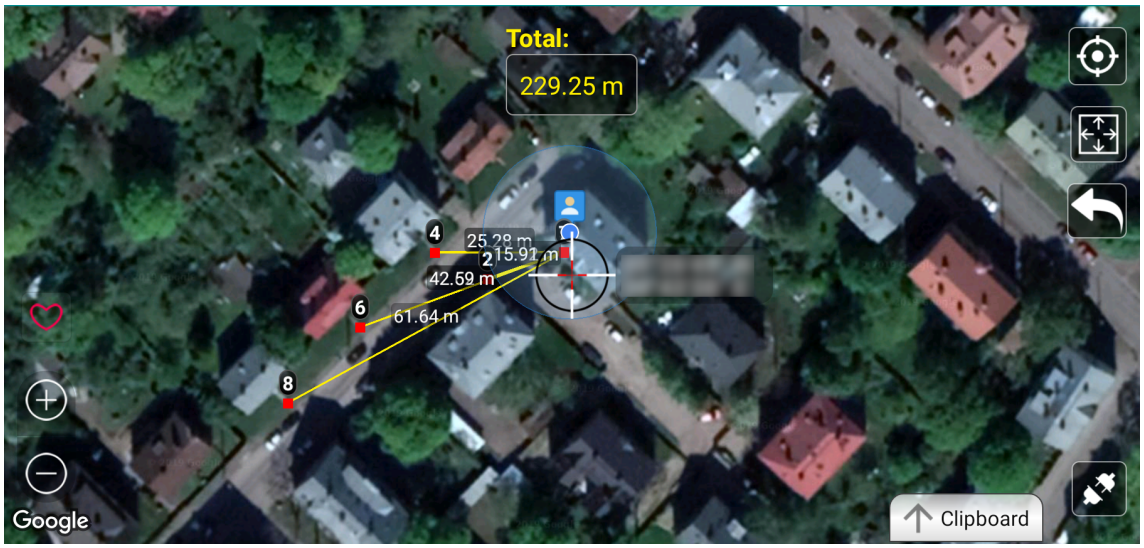

Figure 21. Distance Map for Proximity Experiment.

Experiment Process First, three measurements were made indoors in an Non-Line of Sight (NLOS) position, behind walls or doors. The following measurements were made from the points illustrated on Figure 21.

Experiment Results The following table describes the relation between distance and stuffed beacon (frame) loss. It also illustrates this distance relationship in terms of sent and received frames, time of total transmission and transmission rate.

Table 7. Distance Measurements with Data Transfer.

Distance (M)	Lost Frames	Transfer Rate (kbit/s)	Total Chunks Received	Octets Received	Loss %	Time (s)
0.50	7	46.0085	135	20655	4.93%	7.9681
1.50	2	45.6254	140	21420	1.41%	8.0350
5.00	5	43.8784	137	20961	3.52%	8.3549
15.91	20	43.2577	122	18666	14.08%	8.4748
25.28	6	43.5289	136	20808	4.23%	8.4220
42.62	14	43.5459	128	19584	9.86%	8.4187
61.64	52	46.5494	90	13770	36.62%	7.8755

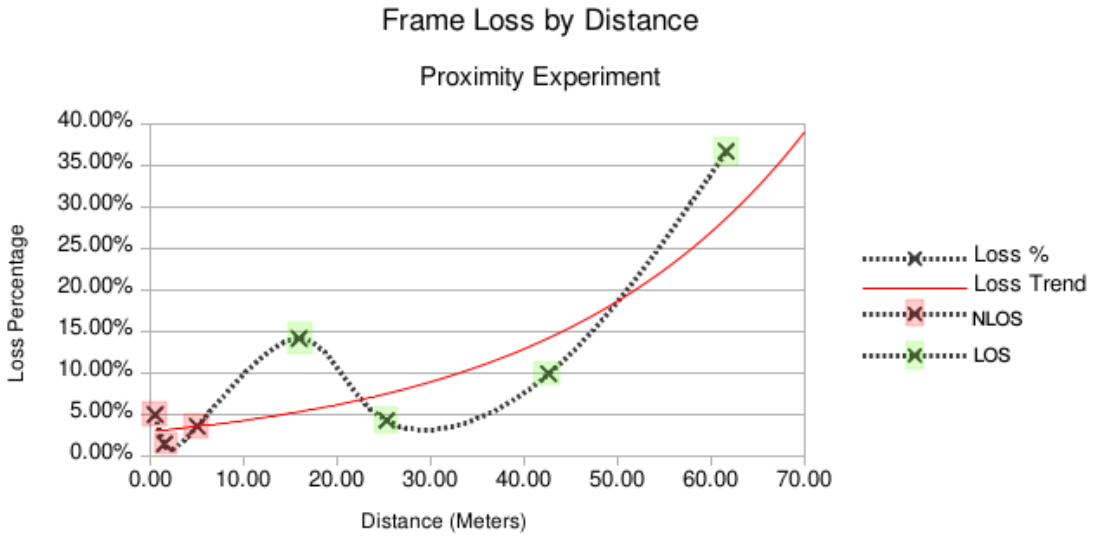

Figure 22. Frame Loss by Distance Trend.

Conclusion These results are expected, with more noise on the channel, information loss on the covert channel happens; with more distance between target and attacker, more

chunks are lost. This makes the chunk re-transmission functions described in 3.5.2 crucial for a successful attack. In our experiments, even at LOS short distance, frame loss ocurred. This proves the need for a basic QoS function, which was implemented in the artifact as described in 3.5.2. Considering the re-transmission functionality, the attack feasibility as described in the attack scenarios on 3.1.2 is feasible. The attack can be done successfully from LOS at a distance of around 30 meters. For expanding the feasible distance, an attacker could use a stronger WiFi element and a directional antenna, yielding better results than the setup described in this experiment.

4.7 Evaluation Discussions

The proposed solution reaches the objectives defined at the beginning of this document, however, some limitations apply depending on hardware and platform the tool is being run on, as demonstrated by our experiments.

Rogue AP detectors analyzing physical properties of the signal, or correlating between the information contained in the beacon frame Elements would be able to detect stuffed beacons as ”Fake AP attack”, incorrectly categorizing the covert channel attack.

The beacon stuffing attack is easily detectable by our artifact if the attacker uses reserved elements to hide the information, regardless if its encrypted or not. In its current version, beaconLeak exploits this fact to generate logs when it detects the usage of reserved elements. Additional IoCs to imporve detection were also developed, evaluated and presented.

Hiding encrypted information in non-reserved elements, maintaining a small beacon frame size and cloning a beacon from the proximity will make detection harder. New techniques would be needed to detect this type of beacon stuffing. Furthermore, using probe frames instead of beacon frames would make this covert channel undetectable to current tools.

In the course of this work it was found that while monitoring was a very reliable function in different operating systems, packet injection was not. The ability to inject packets is necessary to craft stuffed beacons, without it the information will never leave through the WiFi NIC, making data extraction impossible. However, even without packet injection capabilities, an attacker could still use the tool to send execution commands to a target,

albeit without any output or validation of the success of execution. This fact restricts the scope of the attack to a limited set of platform and hardware combinations.

The tool proves that Linux machines in general would be vulnerable to this technique as an exfiltration method, as the Linux kernel is more readily available for packet injection with common wireless NICs, making the attack more feasible on this platform.

Due to the current constraints on packet injection in the Windows platform, i.e. the lack of specific wireless hardware or the need for a customized driver for this attack, makes it unfeasible until packet injection is more readily available on Windows.

Android and Mac OS X, being Linux derivatives, inherit the capabilities needed for this attack method to be feasible. However, some platform and hardware limitations apply so we conclude that the feasibility of the attack on this operating systems is partial.

This attack would be used in scenarios where the attacker is concerned with the later stages of the cyber kill chain [2]. With the work presented in this research, the scenario is further reduced to the following specific limitations:

- Initial foothold with superuser privileges
- Devices running a Linux Kernel
- Devices with a Wireless Network Interface Card that supports both monitor mode and packet injection
- Devices within wireless proximity.
- Non-Default configurations or options in the used attack tools.
- Slow transfer rate to increase covertness.

The speed of transfer is limited to the noise in the wireless channel and the lowest transmission rate from either attacker or victim. This is the key element that would make this method attractive to a threat agent as compared to other methods. In response to this, the detection mechanism simply checks for beacon size, effectively cropping the undetected range of frame size around 350 bytes per frame, resulting in transfer rates capped at around 45 kilo bits per second. As mentioned in their study about covert channels, Giani et. al. make a clear relationship between transmission rate and covertness [33]:

$$\text{Covertness} \propto (\text{Capacity of the Medium} - \text{Transmission Rate})$$

Figure 23. Covertness and Transmission Proportionality as defined in [33, p. 620103-3].

This means that, when detectable, lower transmission rates increase covertness. Without a valid detection mechanism, this technique was not restricted by this relationship. Our artifact is now aware of this and thus has decreased the feasibility of this technique for data exfiltration dut to lower transmission rate.

On the issue of air gaps, this method works if the previously mentioned limitation are met and if the air gapped system is left with the WiFi NIC on it. Considering that this method happens at the physical management of wireless network, layer 2, actual connection or association is never needed. We find it a stretch though that devices left with their wireless network cards would be considered properly in an air gap.

This attack is feasible as presented in our scenarios, specially because results of our experiments show that it is viable to have this attack working from medium ranges, outside premises. A distance of 30 meters, LOS or NLOS, to a victim in addition to at least a 500mW 802.11 radio, will be able to carry out the attack. This could be a vehicle in the parking lot or a neighboring building. The main challenge to the attack is signal noise, as this causes beacons to be lost. However, the implemented re-transmission capabilities as basic QoS gives the covert channel more resilience.

As the attack method is similar to the detection method in their communication mechanism, we can safely assume that detection rate is as high as the effectiveness of the communication channel between the attacker and victim is. This means that monitors in general will work better if close to a victim to certainly detect reception or transmission of stuffed beacons. However, proximity test show that on an attack session, enough traffic will be generated that at least partial stuffed beacons would be caught, enough to log the event. Ideally the detection sensor should have a powerful receptor to maximize its effectiveness.

4.8 Future Work

From the work presented, these are some ideas that would fill some gaps or extend the functionality of the proposed solution.

4.8.1 Detection

Encryption Detection As evaluated, an average size beacon that is not using any default options from tools which hides the information on valid Elements, such as Element 221 for Vendor Specific Information, would be undetected by our artifact. We can extend the detection mechanism to check for encrypted content on the frame element, which could be used as a possible IoC for stuffed beacons.

Physical Signal Detection Implementing something similar to Passive Radiometric Device Identification System (PARADIS) [42], for rogue signal detection using software defined radio solutions like the HackRF. PARADIS has the advantage that it works on even a lower layer, the physical layer. By setting a baseline of the radios allowed to transmit information in the vicinity of the detector, unregistered transmissions should be flagged as possible IoCs. According to the classification proposed by Sobh [34], this would be adding anomaly detection to our artifact.

Machine Learning Baselines Currently a WiFi radio is limited to either a single channel for detection or using a channel hopping technique which lowers detection rate. To solve this, a device nicknamed ”WiFi Cactus” could be used, as it has a single dedicated WiFi device for each channel in both 2.4Ghz and 5Ghz spectrums [54]. Another option would be to use a Software Defined Radio (SDR) device to cover the whole breath of wireless channels. With these devices, we can then use a baseline approach to determine common beacons in the surrounding area, using machine learning algorithms to detect anomalies and report them. This functionality could serve as an extra to Rogue AP detection, allowing for the commercialization of the software and possible hardware sensors.

4.8.2 Attack

Rogue AP Evasion Due to the functionality description of Rogue AP detectors like Kao’s [39], Brik’s [42] and sentrygun [40], it was assumed that these type of sensors would detect stuffed beacons as ”fake ap” or ”evil twin” attack. An experiment should be done with our current tool to check if that is the case. A future version of beaconLeak should include an option, similar to the already implemented "covert" mode, that would set the covert channel using stuffed probes instead. This would keep the covert channel

out of the scope of Rogue AP detectors and similar implementations of WIDS.

5GHz Spectrum The scope of this work happened on the 2.4GHz spectrum of the WiFi specification. Currently the 5GHz spectrum remains unused and undetected by the presented artifact. It would make sense to expand the capabilities to these frequencies too.

Network Tunnel Networking tunnel using Unix sockets to bridge different network stacks over WiFi beacon frames covert communication. This means mounting the network stack on the management of this medium, which would not be an easy task but would exponentially expand the capabilities of the covert channel.

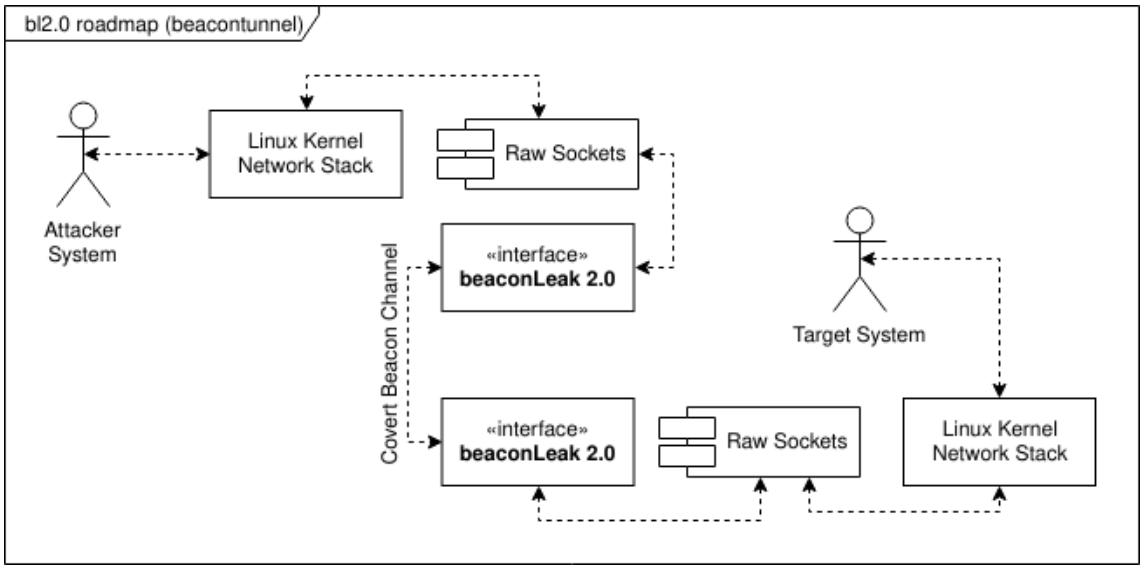

Figure 24. Planned beaconLeak 2.0 Network Tunnel.

5 Conclusion

Following the DSRM process [25], this master thesis deals with the communication of the designed artifact.

In this work, we followed the Design Science Research Methodology process to identify a problem and motivation, set our objectives, design and develop an artifact, evaluate the artifact, and demonstrate this design artifact in an iterative manner.

This design artifact fulfills the objectives we set at the introduction of this research:

- Developed and evaluated a tool that proves the viability of beacon stuffing as an attack method for the later stages of the cyber kill chain.
- Developed and evaluated the capability of this tool to also detect and generate indicators of compromise for security correlation and monitoring tools to consume.
- Evaluated different aspects of the implementation to further prove the viability of beaconLeak, our design artifact, as both an attack and defense solution.

Furthermore, we successfully answer the research question asked in the introduction of this research, along with the extra questions. **Can we use the WiFi beacon frame stuffing technique as a covert channel?**

- Can we extract confidential data through this covert channel?
- Is this covert channel detectable?
- In which scenarios could this method be used?
- In which platforms would this attack be feasible?
- Can Rogue AP detectors be a solution against this attack method or covert channel?
- What is the speed of transfer of this covert channel?
- If this is detectable, what would be valid indicators of compromise?
- Will this method work on air-gapped systems?
- What is the practical physical proximity for this technique be usable?

The beacon stuffing method can be used as a covert channel to exfiltrate confidential data. This covert channel is detectable, we provide a novel tool to detect it and generate system logs for other tools to consume. We analyzed and presented two general scenarios where this method could be used, from an insider threat and from an external threat. An evaluation of the platforms where this attack would be feasible to run was presented. We assume that advanced Rogue AP detectors could detect this tool, specifically those that correlate the information on the Elements or base their detection on the physical properties of the signal, and thus propose future work to both evade this detectors and add new indicators to our detector based on these evasions.

We evaluated the speed of transfer before and after developing better detection indicators and found that the covert channel needs to use a size for each beacon frame transmitted fixed to under 351 bytes, in order to avoid our detector, which results in data transfers of approximately 45 kilo bits per second. The design and evaluation of several indicators of compromise to detect this attack method were presented, the main indicator being the size of the beacon frame. We assume that this method would work on air gap systems as long as they run a Linux OS and still have their WiFi NIC in them. Finally, we evaluated a practical physical proximity for this attack to be approximately 30 meters at a NLOS.

The code to this tool has been published under an open source license and can be found in the following link: `https://github.com/cjcase/beaconleak`

References

[1] Mandiant, "Apt1 - exposing one of chinas cyber espionage units." https://www.fireeye.com/content/dam/fireeye-www/services/pdfs/mandiant-apt1-report.pdf, 2013.

[2] R. M. A. Eric M. Hutchins, Michael J. Cloppert, "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill-chains," in *Proceedings of the Sixth International Conference on Information Warfare and Security*, p. 113125, 2010.

[3] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[4] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[5] K. S. Yim, "The rowhammer attack injection methodology," in *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*, pp. 1–10, Sep. 2016.

[6] T. Vateva-Gurova and N. Suri, "On the detection of side-channel attacks," in *2018 IEEE 23rd Pacific Rim International Symposium on Dependable Computing (PRDC)*, pp. 185–186, Dec 2018.

[7] M. Guri, M. Monitz, Y. Mirski, and Y. Elovici, "Bitwhisper: Covert signaling channel between air-gapped computers using thermal manipulations," in *2015 IEEE 28th Computer Security Foundations Symposium*, pp. 276–289, July 2015.

[8] M. Guri, Y. A. Solewicz, A. Daidakulov, and Y. Elovici, "Fansmitter: Acoustic data exfiltration from (speakerless) air-gapped computers," *CoRR*, vol. abs/1606.05915, 2016.

[9] M. Guri, B. Zadov, A. Daidakulov, and Y. Elovici, "xled: Covert data exfiltration from air-gapped networks via router leds," *CoRR*, vol. abs/1706.01140, 2017.

[10] M. Guri, Y. A. Solewicz, A. Daidakulov, and Y. Elovici, "Diskfiltration: Data exfiltration from speakerless air-gapped computers via covert hard drive noise," *CoRR*, vol. abs/1608.03431, 2016.

[11] M. Guri, M. Monitz, and Y. Elovici, "Usbee: Air-gap covert-channel via electromagnetic emission from usb," in *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pp. 264–268, Dec 2016.

[12] M. Guri, B. Zadov, E. Atias, and Y. Elovici, "Led-it-go: Leaking (a lot of) data from air-gapped computers via the (small) hard drive LED," *CoRR*, vol. abs/1702.06715, 2017.

[13] M. Guri, Y. A. Solewicz, A. Daidakulov, and Y. Elovici, "MOSQUITO: covert ultrasonic transmissions between two air-gapped computers using speaker-to-speaker communication," *CoRR*, vol. abs/1803.03422, 2018.

[14] M. Guri, D. Bykhovsky, and Y. Elovici, "air-jumper: Covert air-gap exfiltration/infiltration via security cameras & infrared (IR)," *CoRR*, vol. abs/1709.05742, 2017.

[15] "Ieee standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements - part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications," *IEEE Std 802.11-2016 (Revision of IEEE Std 802.11-2012)*, pp. 1–3534, Dec 2016.

[16] R. Chandra, J. Padhye, L. Ravindranath, and A. Wolman, "Beacon-stuffing: Wi-fi without associations," in *Eighth IEEE Workshop on Mobile Computing Systems and Applications*, pp. 53–57, March 2007.

[17] V. Gupta and M. K. Rohil, "Bit-stuffing in 802. 11 beacon frame: Embedding non-standard custom information," *International Journal of Computer Applications*, vol. 63, no. 2, 2013.

[18] S. Zehl, N. Karowski, A. Zubow, and A. Wolisz, "Lows: A complete open source solution for wi-fi beacon stuffing based location-based services," in *2016 9th IFIP Wireless and Mobile Networking Conference (WMNC)*, pp. 25–32, July 2016.

[19] Z. Rejiba, X. Masip-Bruin, and E. Marín-Tordera, "Analyzing the deployment challenges of beacon stuffing as a discovery enabler in fog-to-cloud systems," in *2018*

European Conference on Networks and Communications (EuCNC), pp. 1–276, June 2018.

[20] T. Neaves, "Smuggler - an interactive 802.11 wireless shell without the need for authentication or association." in Trustwave SpiderLab's Blog, November 2014, https://www.trustwave.com/Resources/SpiderLabs-Blog/Smuggler—An-interactive-802-11-wireless-shell-without-the-need-for-authentication-or-association.

[21] Y. Nativ, "Pyexfil: A python package for data exfiltration." https://github.com/ytisf/PyExfil.

[22] H. Jiang, "The map of cybersecurity domains (version 2.0)." https://www.linkedin.com/pulse/map-cybersecurity-domains-version-20-henry-jiang-ciso-cissp, 2017.

[23] W. P. Delaney, R. G. Atkins, A. D. Bernard, D. M. Boroson, D. J. Ebel, A. Feder, J. G. Fleischman, M. P. Shatz, R. Stein, and S. D. Weiner, *Systems Analysis and Red Teaming*. MITP, 2015.

[24] V. Mavroeidis and S. Bromander, "Cyber threat intelligence model: An evaluation of taxonomies, sharing standards, and ontologies within cyber threat intelligence," in *2017 European Intelligence and Security Informatics Conference (EISIC)*, pp. 91–98, Sep. 2017.

[25] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *Journal of Management Information Systems*, vol. 24, no. 3, pp. 45–77, 2007.

[26] R. A. S. Gonçalves, "A mac layer covert channel in 802.11 networks," 2011.

[27] K. Sawicki and Z. Piotrowski, "The proposal of ieee 802.11 network access point authentication mechanism using a covert channel," in *2012 19th International Conference on Microwaves, Radar Wireless Communications*, vol. 2, pp. 656–659, May 2012.

[28] F. Mandiant, "M-trends 2019 special report." https://www.fireeye.com/current-threats/apt-groups.html, 2019.

[29] W. Bryant, "International conflict and cyberspace superiority: Theory and practice," *International Conflict and Cyberspace Superiority: Theory and Practice*, pp. 1–239, 01 2015.

[30] T. M. Chen and S. Abu-Nimeh, "Lessons from stuxnet," *Computer*, vol. 44, pp. 91–93, April 2011.

[31] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security Privacy*, vol. 9, pp. 49–51, May 2011.

[32] R. A. Kemmerer, "Shared resource matrix methodology: An approach to identifying storage and timing channels," *ACM Transactions on Computer Systems*, 1983.

[33] G. V. C. Annarita Giani, Vincent H. Berk, "Data exfiltration and covert channels," 2006.

[34] T. S. Sobh, "Wired and wireless intrusion detection system: Classifications, good characteristics and state-of-the-art," *Computer Standards & Interfaces*, vol. 28, no. 6, pp. 670–694, 2006.

[35] S. Zaman and F. Karray, "Tcp/ip model and intrusion detection systems," in *2009 International Conference on Advanced Information Networking and Applications Workshops*, pp. 90–96, May 2009.

[36] S. Boob and P. Jadhav, "Wireless intrusion detection system," *International Journal of Computer Applications*, vol. 5, no. 8, pp. 9–13, 2010.

[37] J. Abo Nada and M. Rasmi Al-Mosa, "A proposed wireless intrusion detection prevention and attack system," in *2018 International Arab Conference on Information Technology (ACIT)*, pp. 1–5, Nov 2018.

[38] B. Hauer, "Data and information leakage prevention within the scope of information security," *IEEE Access*, vol. 3, pp. 2554–2565, 2015.

[39] K. F. Kao, W. C. Chen, J. C. Chang, and H. T. Chu, "An accurate fake access point detection method based on deviation of beacon time interval," in *2014 IEEE Eighth International Conference on Software Security and Reliability-Companion*, pp. 1–2, June 2014.

[40] s0lst1c3, "sentrygun: Rogue ap killer." https://github.com/s0lst1c3/sentrygun, 2017.

[41] R. A. S. Gonçalves, "Rogueap-detector: Rogue access point detector." https://github.com/anotherik/RogueAP-Detector, 2019.

[42] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless device identification with radiometric signatures," in *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*, MobiCom '08, (New York, NY, USA), pp. 116–127, ACM, 2008.

[43] R. Matuleviçius, *Fundamentals of Secure System Modelling*. Springer International Publishing, 2017.

[44] P. L. Philippe Biondi, Guillaume Valadon, "Scapy: the python-based interactive packet manipulation program and library.." https://github.com/secdev/scapy.

[45] P. C. Authority, "pynacl - python binding to the networking and cryptography (nacl) library." https://github.com/pyca/pynacl.

[46] F. Denis, "libsodium - a modern, portable, easy to use crypto library." https://github.com/jedisct1/libsodium.

[47] P. S. Daniel J. Bernstein, Tanja Lange, "Nacl: Networking and cryptography library." https://nacl.cr.yp.to/.

[48] P. C. Authority, "Pynacl: Python binding to the libsodium library." [Documentation] https://pynacl.readthedocs.io.

[49] D. J. Bernstein, T. Lange, and P. Schwabe, "The security impact of a new cryptographic library," in *Proceedings of the 2Nd International Conference on Cryptology and Information Security in Latin America*, LATINCRYPT'12, (Berlin, Heidelberg), pp. 159–176, Springer-Verlag, 2012.

[50] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky, "Comparing the usability of cryptographic apis," in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 154–171, May 2017.

[51] P. Syverson, "A taxonomy of replay attacks [cryptographic protocols]," in *Proceedings The Computer Security Foundations Workshop VII*, pp. 187–191, June 1994.

[52] TP-Link, "tp-link tl-wn7200nd technical specifications." https://www.tp-link.com/us/home-networking/usb-adapter/tl-wn7200nd/, 2019.

[53] Intel, "Intelő centrinoő advanced-n 6205, dual band product specifications." https://ark.intel.com/content/www/us/en/ark/products/59471/intel-centrino-advanced-n-6205-dual-band.html, 2019.

[54] darkmatter, "The hashtag wifi-cactus (wificactus def con 25)." https://palshack.org/the-hashtag-wifi-cactus-wificactus-def-con-25/, August 2017.

Appendix 1 – Beacon Size Experiment Script

```
#!/usr/bin/env python
#
# beaconLeak - Beacon Size Measurements

banner = """
      _                            __              _
     | |_ ___ ___ ___ ___ ___|  |    ___ ___| |_
     | . | -_| .'|  _| . |    |  |__| -_| .'| '_|
     |___|___|__,|___|___|_|_|_____|___|__,|_,_|
                              by cjcase [test]\n
"""

import os
import sys
import time
from datetime import date
import subprocess

from scapy.all import Dot11Beacon
from scapy.all import sniff
from scapy.utils import PcapWriter

# globals
global if0, b_stats, beacons, science

# interface
if0 = "wlan0mon"

# stats
b_stats = []
science = {}
beacons = {}

def do_science(pkt):
    global beacons
    try:
        beacons[pkt.addr2] += 1
    except KeyError as e:
        beacons[pkt.addr2] = 1

def beacon_histogram():
    import pprint, json
    global beacons, science
    channels = range(1,14)
```

```
45      start = time.time()
46      gmt = time.strftime('%d.%m.%Y %H:%M:%S', time.gmtime(
            start))
47      f_prefix = time.strftime('%d.%m.%Y-%H:%M:%S', time.gmtime
            (start))
48      print("[*] start time: {} GMT".format(gmt))
49      # clear data
50      beacons = {}
51      for ch in channels:
52          print("\t[i] channel: ", ch)
53          subprocess.run("iw dev {} set channel {}".format(if0,
                ch), shell=True)
54          pkts = sniff(iface=if0,
55              lfilter=lambda pkt: pkt.haslayer(Dot11Beacon),
56              prn=do_science,
57              timeout=(60*1) # 1 minute
58          )
59          b_uniq = len(beacons.keys())
60          b_total = sum(beacons.values())
61          p_total = len(pkts)
62          if p_total > 0:
63              b_size_avg = sum([len(p) for p in pkts]) / float(
                    p_total)
64          science[ch] = {'b_uniq':b_uniq, 'b_size_avg':
                b_size_avg, 'b_total':b_total}
65          print("\t[i] unique beacons: {}, size avg: {}, total
                beacons: {} (sanity:{})".format(b_uniq, b_size_avg
                , b_total, b_total==p_total))
66          n_pcap = '{}-ch{}.pcap'.format(f_prefix, ch)
67          f_pcap = PcapWriter(n_pcap)
68          f_pcap.write(pkts)
69          print("\t[i] wrote packets to: {}\n".format(n_pcap))
70          #cleanup
71          beacons = {}
72          b_size_avg = 0
73          f_pcap.close()
74      print("[*] results:\n")
75      pprint.pprint(science)
76      f_report = '{}-science.json'.format(f_prefix)
77      f = open(f_report, 'w+')
78      f.write(json.dumps(science))
79      print("[*] results saved to file: {}".format(f_report))
80      end = time.time()
81      print("[*] end of experiment, duration: {} seconds".
            format(end - start))
82
```

```
83 if __name__ == '__main__':
84     print(banner)
85     beacon_histogram()
```

Appendix 2 – Artifact Output

Beacon Size Experiment - Hotel Location Script Output

```
 _                          __            _
| |_ ___ ___ ___ ___ ___|  |    ___ ___| |_
| . | -_| .'|  _| . |   |  |__| -_| .'| '_|
|___|___|__,|___|___|_|_|_____|___|__,|_,_|
                          by cjcase [test]


[*] Beacon Size IoC Experiment
[*] start time: 09.04.2019 23:16:11 GMT
        [i] channel:  1
        [i] unique beacons: 14, size avg: 264.30980683506687,
            total beacons: 6730 (sanity:True)
        [i] wrote packets to: 09.04.2019-23:16:11-ch1.pcap

        [i] channel:  2
        [i] unique beacons: 39, size avg: 264.2990888755524,
           total beacons: 18329 (sanity:True)
        [i] wrote packets to: 09.04.2019-23:16:11-ch2.pcap

        [i] channel:  3
        [i] unique beacons: 6, size avg: 263.117859725235,
           total beacons: 1383 (sanity:True)
        [i] wrote packets to: 09.04.2019-23:16:11-ch3.pcap

        [i] channel:  4
        [i] unique beacons: 8, size avg: 264.39484311898104,
           total beacons: 3219 (sanity:True)
        [i] wrote packets to: 09.04.2019-23:16:11-ch4.pcap

        [i] channel:  5
        [i] unique beacons: 8, size avg: 277.55093555093555,
           total beacons: 3848 (sanity:True)
        [i] wrote packets to: 09.04.2019-23:16:11-ch5.pcap

        [i] channel:  6
        [i] unique beacons: 13, size avg: 265.4672965116279,
           total beacons: 2752 (sanity:True)
        [i] wrote packets to: 09.04.2019-23:16:11-ch6.pcap

        [i] channel:  7
        [i] unique beacons: 13, size avg: 264.4407778432528,
```

```
           total beacons: 3394 (sanity:True)
        [i] wrote packets to: 09.04.2019-23:16:11-ch7.pcap

        [i] channel:  8
        [i] unique beacons: 12, size avg: 264.38456268554467,
            total beacons: 4379 (sanity:True)
        [i] wrote packets to: 09.04.2019-23:16:11-ch8.pcap

        [i] channel:  9
        [i] unique beacons: 7, size avg: 265.78351206434314,
           total beacons: 2984 (sanity:True)
        [i] wrote packets to: 09.04.2019-23:16:11-ch9.pcap

        [i] channel:  10
        [i] unique beacons: 6, size avg: 261.46198830409355,
           total beacons: 171 (sanity:True)
        [i] wrote packets to: 09.04.2019-23:16:11-ch10.pcap

        [i] channel:  11
        [i] unique beacons: 17, size avg: 260.03574303654807,
            total beacons: 6211 (sanity:True)
        [i] wrote packets to: 09.04.2019-23:16:11-ch11.pcap

        [i] channel:  12
        [i] unique beacons: 23, size avg: 264.5677830940989,
           total beacons: 6897 (sanity:True)
        [i] wrote packets to: 09.04.2019-23:16:11-ch12.pcap

        [i] channel:  13
        [i] unique beacons: 48, size avg: 264.349374969119,
           total beacons: 20239 (sanity:True)
        [i] wrote packets to: 09.04.2019-23:16:11-ch13.pcap

[*] results:
{1: {'b_size_avg': 264.30980683506687, 'b_total': 6730, '
   b_uniq': 14},
 2: {'b_size_avg': 264.2990888755524, 'b_total': 18329, '
    b_uniq': 39},
 3: {'b_size_avg': 263.117859725235, 'b_total': 1383, 'b_uniq
    ': 6},
 4: {'b_size_avg': 264.39484311898104, 'b_total': 3219, '
    b_uniq': 8},
 5: {'b_size_avg': 277.55093555093555, 'b_total': 3848, '
    b_uniq': 8},
 6: {'b_size_avg': 265.4672965116279, 'b_total': 2752, '
    b_uniq': 13},
```

```
 7: {'b_size_avg': 264.4407778432528, 'b_total': 3394, '
    b_uniq': 13},
 8: {'b_size_avg': 264.38456268554467, 'b_total': 4379, '
    b_uniq': 12},
 9: {'b_size_avg': 265.78351206434314, 'b_total': 2984, '
    b_uniq': 7},
 10: {'b_size_avg': 261.46198830409355, 'b_total': 171, '
    b_uniq': 6},
 11: {'b_size_avg': 260.03574303654807, 'b_total': 6211, '
    b_uniq': 17},
 12: {'b_size_avg': 264.5677830940989, 'b_total': 6897, '
    b_uniq': 23},
 13: {'b_size_avg': 264.349374969119, 'b_total': 20239, '
    b_uniq': 48}}
```

Beacon Size Experiment - Bank Location Script Output

```

     _                          __             _
    | |_ ___ ___ ___ ___ ___|  |   ___ ___| |_
    | . | -_| .'|  _| . |   |  |__| -_| .'| '_|
    |___|___|__,|___|___|_|_|_____|___|__,|_,_|
                           by cjcase [test]


[*] Beacon Size IoC Experiment
[*] start time: 10.04.2019 06:50:24 GMT
        [i] channel:  1
        [i] unique beacons: 11, size avg: 273.51708766716195,
            total beacons: 3365 (sanity:True)
        [i] wrote packets to: 10.04.2019-06:50:24-ch1.pcap

        [i] channel:  2
        [i] unique beacons: 1, size avg: 318.0, total beacons
           : 473 (sanity:True)
        [i] wrote packets to: 10.04.2019-06:50:24-ch2.pcap

        [i] channel:  3
        [i] unique beacons: 1, size avg: 252.0, total beacons
           : 2 (sanity:True)
        [i] wrote packets to: 10.04.2019-06:50:24-ch3.pcap

        [i] channel:  4
        [i] unique beacons: 1, size avg: 318.0, total beacons
           : 1 (sanity:True)
        [i] wrote packets to: 10.04.2019-06:50:24-ch4.pcap
```

```
26
27          [i] channel:  5
28          [i] unique beacons: 1, size avg: 252.0, total beacons
               : 2 (sanity:True)
29          [i] wrote packets to: 10.04.2019-06:50:24-ch5.pcap
30
31          [i] channel:  6
32          [i] unique beacons: 13, size avg: 270.86701097482245,
                total beacons: 3098 (sanity:True)
33          [i] wrote packets to: 10.04.2019-06:50:24-ch6.pcap
34
35          [i] channel:  7
36          [i] unique beacons: 1, size avg: 252.0, total beacons
               : 3 (sanity:True)
37          [i] wrote packets to: 10.04.2019-06:50:24-ch7.pcap
38
39          [i] channel:  8
40          [i] unique beacons: 0, size avg: 0, total beacons: 0
               (sanity:True)
41          [i] wrote packets to: 10.04.2019-06:50:24-ch8.pcap
42
43          [i] channel:  9
44          [i] unique beacons: 1, size avg: 252.0, total beacons
               : 5 (sanity:True)
45          [i] wrote packets to: 10.04.2019-06:50:24-ch9.pcap
46
47          [i] channel:  10
48          [i] unique beacons: 2, size avg: 234.6, total beacons
               : 45 (sanity:True)
49          [i] wrote packets to: 10.04.2019-06:50:24-ch10.pcap
50
51          [i] channel:  11
52          [i] unique beacons: 5, size avg: 261.14895729890765,
               total beacons: 1007 (sanity:True)
53          [i] wrote packets to: 10.04.2019-06:50:24-ch11.pcap
54
55          [i] channel:  12
56          [i] unique beacons: 2, size avg: 255.5, total beacons
               : 16 (sanity:True)
57          [i] wrote packets to: 10.04.2019-06:50:24-ch12.pcap
58
59          [i] channel:  13
60          [i] unique beacons: 1, size avg: 313.0, total beacons
               : 237 (sanity:True)
61          [i] wrote packets to: 10.04.2019-06:50:24-ch13.pcap
62
```

```
[*] results:

{1: {'b_size_avg': 273.51708766716195, 'b_total': 3365, '
   b_uniq': 11},
 2: {'b_size_avg': 318.0, 'b_total': 473, 'b_uniq': 1},
 3: {'b_size_avg': 252.0, 'b_total': 2, 'b_uniq': 1},
 4: {'b_size_avg': 318.0, 'b_total': 1, 'b_uniq': 1},
 5: {'b_size_avg': 252.0, 'b_total': 2, 'b_uniq': 1},
 6: {'b_size_avg': 270.86701097482245, 'b_total': 3098, '
    b_uniq': 13},
 7: {'b_size_avg': 252.0, 'b_total': 3, 'b_uniq': 1},
 8: {'b_size_avg': 0, 'b_total': 0, 'b_uniq': 0},
 9: {'b_size_avg': 252.0, 'b_total': 5, 'b_uniq': 1},
 10: {'b_size_avg': 234.6, 'b_total': 45, 'b_uniq': 2},
 11: {'b_size_avg': 261.14895729890765, 'b_total': 1007, '
    b_uniq': 5},
 12: {'b_size_avg': 255.5, 'b_total': 16, 'b_uniq': 2},
 13: {'b_size_avg': 313.0, 'b_total': 237, 'b_uniq': 1}}
[*] results saved to file: 10.04.2019-06:50:24-science.json
[*] end of experiment, duration: 782.0989291667938 seconds
```

Beacon Size Experiment - Residential Location Script Output

```

     _                          __            _
    | |_ ___ ___ ___ ___ ___|  |   ___ ___| |_
    | . | -_| .'|  _| . |   |  |__| -_| .'| '_|
    |___|___|__,|___|___|_|_|_____|___|__,|_,_|
                           by cjcase [test]


[*] Beacon Size IoC Experiment
[*] start time: 12.04.2019 21:22:01 GMT
        [i] channel:  1
        [i] unique beacons: 9, size avg: 255.7782738095238,
           total beacons: 2016 (sanity:True)
        [i] wrote packets to: 12.04.2019-21:22:01-ch1.pcap

        [i] channel:  2
        [i] unique beacons: 2, size avg: 341.74110671936756,
           total beacons: 506 (sanity:True)
        [i] wrote packets to: 12.04.2019-21:22:01-ch2.pcap

        [i] channel:  3
        [i] unique beacons: 1, size avg: 211.0, total beacons
           : 7 (sanity:True)
```

```
21          [i] wrote packets to: 12.04.2019-21:22:01-ch3.pcap
22
23          [i] channel:  4
24          [i] unique beacons: 1, size avg: 296.0, total beacons
               : 1 (sanity:True)
25          [i] wrote packets to: 12.04.2019-21:22:01-ch4.pcap
26
27          [i] channel:  5
28          [i] unique beacons: 1, size avg: 244.0, total beacons
               : 6 (sanity:True)
29          [i] wrote packets to: 12.04.2019-21:22:01-ch5.pcap
30
31          [i] channel:  6
32          [i] unique beacons: 4, size avg: 252.22706766917292,
               total beacons: 1995 (sanity:True)
33          [i] wrote packets to: 12.04.2019-21:22:01-ch6.pcap
34
35          [i] channel:  7
36          [i] unique beacons: 3, size avg: 261.0247524752475,
               total beacons: 606 (sanity:True)
37          [i] wrote packets to: 12.04.2019-21:22:01-ch7.pcap
38
39          [i] channel:  8
40          [i] unique beacons: 1, size avg: 272.0, total beacons
               : 94 (sanity:True)
41          [i] wrote packets to: 12.04.2019-21:22:01-ch8.pcap
42
43          [i] channel:  9
44          [i] unique beacons: 1, size avg: 244.0, total beacons
               : 2 (sanity:True)
45          [i] wrote packets to: 12.04.2019-21:22:01-ch9.pcap
46
47          [i] channel:  10
48          [i] unique beacons: 2, size avg: 170.0, total beacons
               : 2 (sanity:True)
49          [i] wrote packets to: 12.04.2019-21:22:01-ch10.pcap
50
51          [i] channel:  11
52          [i] unique beacons: 3, size avg: 201.51084337349397,
               total beacons: 1245 (sanity:True)
53          [i] wrote packets to: 12.04.2019-21:22:01-ch11.pcap
54
55          [i] channel:  12
56          [i] unique beacons: 0, size avg: 0, total beacons: 0
               (sanity:True)
57          [i] wrote packets to: 12.04.2019-21:22:01-ch12.pcap
```

```
58
59          [i] channel:  13
60          [i] unique beacons: 1, size avg: 275.0, total beacons
               : 273 (sanity:True)
61          [i] wrote packets to: 12.04.2019-21:22:01-ch13.pcap
62
63 [*] results:
64
65 {1: {'b_size_avg': 255.7782738095238, 'b_total': 2016, '
      b_uniq': 9},
66  2: {'b_size_avg': 341.74110671936756, 'b_total': 506, '
       b_uniq': 2},
67  3: {'b_size_avg': 211.0, 'b_total': 7, 'b_uniq': 1},
68  4: {'b_size_avg': 296.0, 'b_total': 1, 'b_uniq': 1},
69  5: {'b_size_avg': 244.0, 'b_total': 6, 'b_uniq': 1},
70  6: {'b_size_avg': 252.22706766917292, 'b_total': 1995, '
       b_uniq': 4},
71  7: {'b_size_avg': 261.0247524752475, 'b_total': 606, 'b_uniq
       ': 3},
72  8: {'b_size_avg': 272.0, 'b_total': 94, 'b_uniq': 1},
73  9: {'b_size_avg': 244.0, 'b_total': 2, 'b_uniq': 1},
74  10: {'b_size_avg': 170.0, 'b_total': 2, 'b_uniq': 2},
75  11: {'b_size_avg': 201.51084337349397, 'b_total': 1245, '
       b_uniq': 3},
76  12: {'b_size_avg': 0, 'b_total': 0, 'b_uniq': 0},
77  13: {'b_size_avg': 275.0, 'b_total': 273, 'b_uniq': 1}}
78 [*] results saved to file: 12.04.2019-21:22:01-science.json
79 [*] end of experiment, duration: 781.8830275535583 seconds
```

Output of beaconleak on 'c2' mode (command and control mode with debugging verbosity)

```
1  root@eve:code # python beaconleak.py --debug --c2c wlan0mon
2
3  _                         __              _
4  | |_ ___ ___ ___ ___ ___|  |   ___ ___| |_
5  | . | -_| .'|  _| . |   |  |__| -_| .'| '_|
6  |___|___|__,|___|___|_|_|_____|___|__,|_,_|
7  by cjcase [v0.7]
8
9
10 [*] Using interface wlan0mon, use Ctrl+C to exit
11 [beaconshell] # uname -a
12 [d] encrypted:
13 b5418dcf74830e2a7dd627cfd734b98be1f454664febfe9f
```

```
fe2d7defcc191e372da45eef9af99241089c6603adf34a4f
[d] scapy frame:
RadioTap()/Dot11(type=0, subtype=8, addr1='ff:ff:ff:ff:ff:ff
   ', addr2='00:14:bf:de:ad:c0', addr3='00:14:bf:de:ad:c0')/
   Dot11Beacon(cap=4352)/Dot11Elt(ID=0, info=b'linksys', len
   =7)/Dot11Elt(ID=1, info=b'\x82\x84\x0b\x16', len=4)/
   Dot11Elt(ID=48, info=b'\x01\x00\x00\x0f\xac\x04\x01\x00\
   x00\x0f\xac\x04\x01\x00\x00\x0f\xac\x02\x0c\x00', len=20)/
   Dot11Elt(ID=253, info=b'\x01', len=1)/Dot11Elt(ID=254,
   info=b"\xb5A\x8d\xcft\x83\x0e*}\xd6'\xcf\xd74\xb9\x8b\xe1\
   xf4TfO\xeb\xfe\x9f\xfe-}\xef\xcc\x19\x1e7-\xa4^\xef\x9a\
   xf9\x92A\x08\x9cf\x03\xad\xf3JO", len=48)
.
Sent 1 packets.
[d] response packet:
RadioTap(version=0, pad=0, len=18, present=18478, notdecoded=
   b'\x10\x02\x99\t\xa0\x00\xec\x01\x00\x00')/Dot11(subtype
   =8, type=0, proto=0, FCfield=0, ID=0, addr1='ff:ff:ff:ff:
   ff:ff', addr2='00:14:bf:de:ad:c0', addr3='00:14:bf:de:ad:
   c0', SC=0, addr4=None)/Dot11Beacon(timestamp=0,
   beacon_interval=100, cap=4352)/Dot11Elt(ID=0, len=7, info=
   b'linksys')/Dot11Elt(ID=1, len=4, info=b'\x82\x84\x0b\x16
   ')/Dot11Elt(ID=48, len=20, info=b'\x01\x00\x00\x0f\xac\x04
   \x01\x00\x00\x0f\xac\x04\x01\x00\x00\x0f\xac\x02\x0c\x00')
   /Dot11Elt(ID=128, len=1, info=b'\x01')/Dot11Elt(ID=254,
   len=134, info=b"\xb4\xc3\xf4\\t\nn\xff\xe1\xab\xe6\x8d\x13
   \xa9V\x17\x0fL\x01\x86\x0fj\xd2\xda\x85\xce]\x9b\xde\xf5\
   x1eB\xb45o\xcf\x07\x9c\x9bD'\x99\x7f&H\x06S\x87].t\xe0E\
   xe3Uh\xe8$W\x9b\xcd\r=%R\xb2\xdb\xba\xcd\x8d\x8ev\x97.lz\
   x078j\x19{\xaa\xcdo\xf9\x12\xa8\x8f~\xff\x8eA\xab\xc6\x8eB
   {\xff\xf5\x9c\xadI\x81\xe6\x8c\xc1S\xe2\x9b\x81VI \x15\xc2
   \x9b\xa9\x84\xb9\xa2\xb0\x14/,\x11W\x9de@\xbf'\x8f\x82\xa1
   ")/Dot11Elt(ID=255, len=4, info=b'\x83\xe4M\x1c')
[d] received:
b4c3f45c740a6effe1abe68d13a956170f4c01860f6ad2d
a85ce5d9bdef51e42b4356fcf079c9b4427997f26480653
875d2e74e045e35568e824579bcd0d3d2552b2dbbacd8d8
e76972e6c7a07386a197baacd6ff912a88f7eff8e41abc6
8e427bfff59cad4981e68cc153e29b8156492015c29ba98
4b9a2b0142f2c11579d6540bf278f82a1

Linux alice 4.19.4-arch1-1-ARCH #1 SMP PREEMPT Fri Nov 23
   09:06:58 UTC 2018 x86_64 GNU/Linux

[beaconshell] # ^C
[*] Done!
```

Appendix 3 – Installation Manuals

Android User Manual - Markdown

```
# beaconLeak on android (root needed)

## Pre-requisites

You will need a **rooted android device**.
If your android device WiFi doesn't support monitor mode, you can use an USB OTG cable to connect an external WiFi card that supports it.

## Dependencies

* termux
  * tsu
  * git
  * clang
  * python-dev
  * libsodium-dev
  * libffi-dev
  * cmake
  * python
    * scapy==2.4.0


## Installation

Run as normal user:

```bash
pkg update
pkg install tsu git clang python python-dev libsodium-dev libffi-dev cmake
tsu
```

Allow root access and then run the following:

```bash
mkdir bl
cd bl
python -m venv blenv
source blenv/bin/activate
pip install scapy==2.4.0
```
```

```
git clone https://github.com/pyca/pynacl
cd pynacl
# git checkout cf132ab (if current build does not work)
find . -type f -not -path '*/\.*' -exec sed -i 's%/bin/sh%/
   data/data/com.termux/files/usr/bin/sh%g' {} \;
python setup.py install
exit
```

if python setup fails, remove the pynacl git repo, clone again and use the git checkout to pull the tested commit.

Back as normal user, install iw and optionally aircrack (for using airmon-ng)

```bash
pkg install root-repo
pkg install iw ethtool aircrack-ng
```

get root, load the virtual env again, set up monitor mode and run:

```bash
tsu
source blenv/bin/activate
iw dev wlan0 set type monitor # OR airmon-ng start wlan0
python beaconleak.py --c2 wlan0mon
```

Setting monitor mode

beaconleak needs a wifi card in monitor mode to operate correctly, if your device has one, you need to first set it in monitor mode for it to work

Experimental

There might be a chance that you can run beaconleak on any non-rooted device if you have an external wifi card with the RTL8187 chipset, this however is untested. Check the following link for more info: https://www.kismetwireless.net/static/android-pcap/

For the adventurous:
https://null-byte.wonderhowto.com/how-to/android-cyanogenmod-

```
   kernel-building-monitor-mode-any-android-device-with-
   wireless-adapter-0162943/
76
77 ## Resources
78
79 https://wiki.termux.com/wiki/
   Instructions_for_installing_python_packages
```

Windows User Manual - Markdown

```
1  # beaconLeak on Windows
2
3  ## Dependencies
4
5  * Python 3
6  * NPCap driver
7
8  ## On Packet Injection
9
10 >*"It is important to realize that at this point in time,
   Windows can only be used for listening to network traffic.
    Windows cannot inject data packets. This is a fundamental
    limitation."* - Aircrack-ng Wiki
11
12 Both beaconLeak and Scapy assume packet injection
   capabilities by default. However, the capability to do so
   resides in both the driver and the firmware of the
   wireless NIC.
13 During the development and research of this tool, no viable
   technique was found to make packet injection possible on
   Windows. While not impossible, further research must be
   done to enable packet injection in a reliable way.
14 **Without packet injection capabilities, beaconLeak will only
    work in Detection mode (both offline and live) and
   partially in leak mode receiveing C2 commands. C2 mode
   will not be available.**
15
16 ## Installation
17
18 Download the latest Python 3 binary installer for AMD64, run
   it with the option to "Add Python 3.x to PATH".
19 Verify that it was correctly added to path by installing
   beaconleak dependencies with pip:
20
21 ```
22 pip install -r requirements.txt
```

```
```

Download the latest Ncap binary installer.
In the NPCap installation, make sure to select "Support raw 802.11 traffic (and monitor mode) for wireless adapters", this will add support for monitor mode.

Alternative: Download the latest Wireshark x64 binary. When installing wireshark, select the option "Install Npcap 0.99-r9".

WlanHelper.exe will be installed along, the utilities will be saved in ```%WINDIR%\System32\Npcap```, so it is a good idea to add this to Window's PATH.

Before you can use WlanHelper to set monitor mode on, you will need to install the Visual C++ Redistributable for Visual Studio 2013.
Run the following in an administrator command to verify its correctly installed:

```
WlanHelper -i
```

Usage

Use WlanHelper.exe to set the desired card in monitor mode, you can then run beaconLeak with the interface name as the example shows:

```
python beaconleak.py --mon "Intel(R) Centrino(R) Advanced-N 6205"
```

Appendix 4 – Cryptography Evidence

Default Key

```
Python 3.7.2 (default, Jan 10 2019, 23:51:51)
[GCC 8.2.1 20181127] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import nacl.utils
>>> import nacl.secret
>>> key = "299e29a4d36990bc479d6fed6551a94c7e3da6e10c8cdf9bab9e3c18a04ddee8"
>>> enc = "b4c3f45c740a6effe1abe68d13a956170f4c01860f6ad2da85ce5d9bdef51e42b4356fcf079c9b4427997f26480653875d2e74e045e35568e824579bcd0d3d2552b2dbbacd8d8e76972e6c7a07386a197baacd6ff912a88f7eff8e41abc68e427bfff59cad4981e68cc153e29b8156492015c29ba984b9a2b0142f2c11579d6540bf278f82a1"
>>> box = nacl.secret.SecretBox(bytes.fromhex(key))
>>> box.decrypt(bytes.fromhex(enc))
b'Linux redb0x 4.19.4-arch1-1-ARCH #1 SMP PREEMPT Fri Nov 23 09:06:58 UTC 2018 x86_64 GNU/Linux\n'
```

Custom Key

```
Python 3.7.3 (default, Mar 26 2019, 21:43:19)
[GCC 8.2.1 20181127] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import nacl.pwhash
>>> import nacl.secret
>>> enc = "0a80de70ab7fb38391882a2ca1ece87f2f9bd19b26ae530274e9db43e51e70436955469b6e40fd086b5455bcad29fc28"
>>> salt = b'beaconleak::salt'
>>> key = nacl.pwhash.argon2i.kdf(32, b'secret', salt)
>>> key
b'\xc2b\x1d\xb00\xf3\xf8H\xa5~\xbf\xa2\xe3d\xda?\xde\x90\x83\xc8\xaffb\xe5D\x95\xe3.\t\x94\xe4\xe2'
>>> box = nacl.secret.SecretBox(key)
>>> box.decrypt(bytes.fromhex(enc))
b'uname -a'
>>>
```