Embedding Data in 802.11 Beacons for Covert Communication

Eric Roch[1]

***Abstract*— Wireless communications are necessarily observable by anyone within range of the transmitter, making covert wireless communication difficult to conceal. This paper explores a technique known as Beacon Stuffing, especially as it relates to covert communication in a wireless network, as well as other applications where low-overhead small-data transmissions are required.**

I. INTRODUCTION

At times, information secrecy can be just as important as confidentiality. Encryption can provide confidentiality – unauthorized individuals will be unable to interpret any information they collect – though it does not provide secrecy, anyone may still monitor the communication. In times when the simple act of communicating can be as revealing as the information being shared, it is important that there is no easy way to locate communicating devices. In a wireless network, all communication is visible to any observer. However, secrecy does not imply invisibility - communication may still take place on a public channel if an observer is unable to detect it. The IEEE 802.11 protocol defines several types of network management frames, one of which is a *beacon*, which broadcasts information about the network capabilities of an Access Point (AP). Beacons are a common sight in wireless network traffic analysis, and as such make for a prime candidate for masking transmissions. If done correctly, an observer will not know the difference between standard network management traffic and covert communication.

Wireless clients listen for beacon frames to determine which networks are available to join. To facilitate that decision, beacons contain information about the supported data rates, power levels, and encryption standards. Sometimes it is necessary for AP vendors to include non-standard data specific to the AP. The 802.11 standard provides a mechanism for this by allowing vendor-specific payloads to be attached to management frames.

In this paper, we will explore beacon stuffing – the use of these vendor-specific payloads as a means for transmitting arbitrary data – with a specific focus on applications where secrecy is desired.

II. RELATED WORK

Several other researchers have looked into using beacons as a means for communication. Among these, R. Chandra et al. proposed overloading the Service Set Identifier (SSID), Basic Service Set Identifier (BSSID) and vendor-specific information element (IE) fields [1]. Additionally, Gupta and Rohil proposed utilizing unused bits in the length field of each IE [2]. The following sections will introduce these techniques and discuss the advantages and shortcomings of each.

A. SSID

The SSID is a unique identifier of the wireless network. All devices connected to a network must agree on the SSID. It can be up to 32 alpha-numeric characters, or 32 bytes [3]. R. Chandra et al. proposed using an encoding scheme of a 1 byte message identifier, followed by 1 byte containing a 7-bit fragment counter and a flag indicating more fragments are available, finally followed by the data. This leaves a maximum of 30 bytes available for data, though the available space will realistically be lower as the base SSID will likely include some identifiable string (i.e. ”netXXXXX...”, where XXXXX is the data appended to the original SSID, ”net”). Assuming a beacon interval of about 10ms, this gives a maximum data rate of 24 Kbps.

B. BSSID

The BSSID is a 6 byte field which traditionally represents the media access control (MAC) address of the AP. However, since the AP MAC address is included in the MAC header of the frame as the transmitter address, this field can be set to any value. By establishing a known SSID, devices may check the BSSID field of matching beacon frames to extract the data. Using the same serialization as before, we can fit 4 data bytes into the BSSID field, giving a transmission rate of 3.2 Kbps.

C. Vendor Information Element

The 802.11 standard includes a general specification for AP vendors to include up to 253 bytes of data in a special IE. The format of these information elements is a 3 byte vendor Organizationally Unique Identifier (OUI), a 1 byte vendor-specific type, and up to 251 bytes of data. The IE can either contain the actual OUI with a type to indicate data, or the OUI can be a constant fake value for filtering and the type can be used for either data or providing segmentation information. Again assuming an interval of 10ms, this gives a transmission rate of about 200 Kbps.

D. Information Element Length Field

Gupta and Rohil proposed a new method, utilizing unused bits in the length field of each IE. Each IE type has a predefined maximum length, some of which require fewer than 8 bits to represent, meaning there are unused bits that are always assumed to be zero. Firmware modification on

[1]Secure and Assured Systems Engineering Intern at Draper and undergraduate at the University of Illinois at Urbana-Champaign. Email: eric at roch.us

the client device would extract the data from these extra bits before passing on the un-stuffed frame to upper layers. Based on the table compiled by Gupta and Rohil, this method provides up to 191 bits per frame if every IE is present [2]. This provides a transmission rate of 19.1 Kbps at a 10ms interval. Realistically, this will be lower as not every IE will be present.

III. Comparison of Data Embedding Techniques

The SSID approach is by far the easiest to implement – it requires no modification to the drivers on the client or AP. However, most client devices (such as phones, tablets, laptops, etc.) display the SSID to the user for the purpose of selecting a network, which means the data would be visible to the user. This is not desirable, especially in the context of covert communication, so this approach will not be considered further.

The BSSID offers a promising alternative to the SSID, as it is not client-facing. Two major limitations are that the BSSID field is not always available (e.g. if the source address is a group address, in which case the BSSID is validated), and it is limited to just 6 bytes per frame [2]. Using the BSSID also requires firmware modifications on both the client and AP.

Using the IE length field is an interesting approach and has a considerable advantage in that it utilizes bits that would otherwise go to waste. This means that you can transmit almost 200 bits per frame with no bandwidth cost. The major limitation is that this method is computationally difficult, requiring separate data structures for storing the locations of these extra bits and preprocessing the whole frame before zeroing out the extra data and passing it on to higher layers. Another drawback is that this method requires modification to the firmware to accommodate future changes to the 802.11 standard which introduce new IE.

The vendor IE approach has the advantage of allowing the most data in a single frame of the methods discussed here, as well as having very little computational cost. One drawback is that embedding longer data increases frame size, which could congest the network. However, for the purposes of this paper, we will assume that the data is either short enough to fit in one or two frames, or is not time sensitive and can be buffered until the network is idle. While this method does require firmware modification, the technique is robust and should not be affected by changes to the 802.11 specification.

IV. POTENTIAL APPLICATIONS

There are numerous potential applications for embedding data in beacon frames. We will list a variety of them here, but the main focus of this paper is on applications in which the act of communicating must be concealed.

A. Location based advertisements

Targeted advertising is mutually beneficial to both consumers and marketers. By narrowing the audience to just those who are likely to act on the advertisement, resources can be used more efficiently. Location based advertisements are a good way to target a group of people who will likely be interested in a topic, coupon, etc. However, privacy is a growing concern among users, and using standard location services such as GPS requires either substantial access to the system or an always running application to send location data, neither of which are ideal. Due to the limited range of wireless signals, advertisements served from an AP will only reach devices in the immediate vicinity, thus providing localization without needing to gather user data.

Beacon stuffing can be used to embed advertisement data such that all devices within range of the AP receive the advertisement or coupon. This would, however, require driver modification in order to present the embedded data to a user. [1]

B. Network monitoring/data extraction

Since beacon stuffing requires modification of the firmware on the AP, we will assume that we have complete access to the AP, but not necessarily to the networks broadcast by the AP (e.g. a compromised AP). In such a scenario, we can use beacon stuffing to extract information about a network or specific traffic without showing suspicious traffic. Special software running on the AP could collect and log data about connected clients and traffic patterns, then dump the log (or a summary) via beacon stuffing on request. A network administrator could use this to quickly diagnose problems or gauge real-time channel utilization and other AP parameters. An adversary could use this to gather metadata about a network without ever connecting to it.

C. Low-overhead burst communication

Consider a situation where a group of devices in the same vicinity, but not necessarily on the same network, need to receive information from a central authority at the same time. One solution would be to send a broadcast or multicast TCP message on each network containing target devices. This would require the formation of a distinct data frame for each network, which will increase the overhead as more networks are added. This requires that every device is associated with a network. Alternatively, we can use beacon stuffing to communicate to all of these devices across multiple networks simultaneously, with fixed overhead. In such a scenario, we might set up a WLAN called ”announcements” which each target device would be able to receive beacons from, even if they are associated with another network. By stuffing the beacons for this announcements WLAN, all devices in the area can receive the same message with a fixed overhead of a single beacon frame.

D. Covert communication

In a situation where an agent must receive information from or report back to a supervisor, but the agent cannot risk the discovery of the communication, beacon stuffing can simplify the setup required compared to other secure channels. If the supervisor can gain control of a public AP, the agent can communicate simply by being in range of the

AP without ever connecting or sending any data frames. In this way, someone trying to identify an undercover agent would need special knowledge of the communication system in order to detect that this communication is taking place. Without advance knowledge, this would look just like a typical beacon. Due to the sheer number of these in a short time frame, manually filtering a network capture is infeasible. A scripted search could be performed, but at that point the attacker knows about the communication channel and we would have to fall back on encryption.

V. BEACON FRAME FORMAT

The 802.11 protocol describes a standard format for beacon frames, which are a type of management frame. Each beacon frame consists of a MAC header, body, and frame check sequence (FCS). The MAC header (shown in Fig. 1) contains information such as source and destination addresses, sequence and fragment IDs, and control flags. The FCS is simply an error detection mechanism that signals the recipient to discard damaged frames.

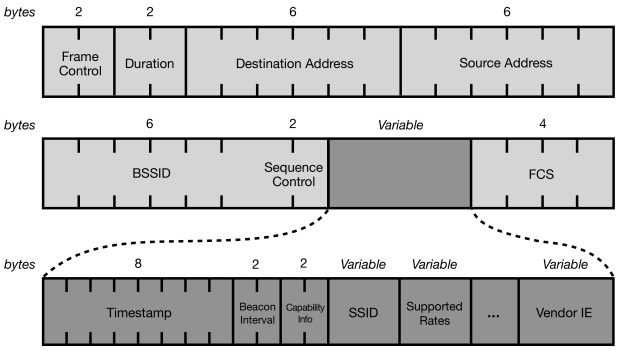

Fig. 1. General structure of 802.11 beacon frame. The MAC header is shown in light gray, the beacon payload is shown in dark gray.

The body of a beacon frame is made up of several required fields and IE, and may contain additional IE. The required fields are timestamp, interval, and capabilities, followed by two IE containing the SSID and supported rates. The body may then contain more IE, up to a total frame size of 2304 bytes. Each IE can be up to 257 bytes, containing a 1 byte ID, 1 byte length field, and N data bytes, where N is the value in the length field. For the vendor IE, the first 4 data bytes are the vendor's OUI and a type specifier, as shown in Fig. 2.

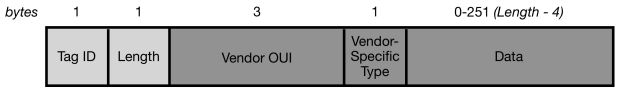

Fig. 2. Structure of Vendor-Specific IE

VI. IMPLEMENTATION

As a proof-of-concept to show that beacon stuffing is feasible, we will explore a simple implementation that utilizes the vendor IE. For sake of brevity and simplicity, we will only consider one AP chipset, the Ralink RT3070L, connected via USB to an Ubuntu host. The only requirement for this demonstration is that the transmitter chipset supports packet injection. This should otherwise be completely general. To view the source code used, visit `https://github.com/emroch/BeaconStuffing`.

A. Data Protection

Some applications of beacon stuffing may require the data to be secure and most will want some form of integrity check to prevent malicious inspection or modification of the data. For this example, we will use a symmetric encryption scheme from the Python Cryptography package known as Fernet encryption, which utilizes AES-128-CBC for encryption and SHA256 based HMAC for integrity [5]. To use this, we encrypt the message before constructing the beacon. This will be discussed more in the Results and Considerations section.

B. Frame Creation

While it does not matter how the frame is actually created (it could be written directly in binary), we will use a Python library called Scapy [4] to assist with frame creation.

To generate a stuffed beacon frame, we first create a basic frame containing the MAC header. We then split the message into chunks of 251 bytes, if necessary, and embed them into vendor IE which are then attached to the base frame as a payload. If there were multiple chunks, we have a fragmented frame, so we will use the MAC header's *fragment ID* field and *More Fragments* flag to indicate to the receiver that the message is fragmented.

C. Message Recovery

To recover the original message, we must first record the transmitted beacons. This is easy to do using Wireshark [6] another computer. By saving a packet capture file which contains our custom beacon, we can use Scapy and Python again to filter the file and recover the message. We start by loading the saved capture file, then apply a filter based on some criteria which are unique to our beacons. This could be an SSID (such as "announcements"), a BSSID, or a special IE. After obtaining a filtered list of just the stuffed packets, we can dissect them to extract the IE containing our message. We will store the extracted message pieces in a dictionary keyed with the fragment ID of the packet, which allows us to reconstruct the original message if the beacons are out of order and to determine if any pieces are missing. Once reassembled, the message can be decrypted and processed.

VII. RESULTS AND CONSIDERATIONS

Using the above procedure, we were successfully package, transmit, receive, and unpack messages both in single beacons and fragmented across multiple frames. There were, however, several considerations that should be taken into account before deploying a similar technique in the field.

1) The encryption scheme used provides both confidentiality and integrity, but relies on a pre-shared key.

This could be problematic for deployments where an AP may have to be set up in advance, running the risk of the shared key being discovered. If encryption is desired, a secure key exchange would increase confidence, at the cost of increased overhead.

2) We did not consider a method for reconstructing messages that have missing fragments. Since each fragment has a unique ID, it would be possible to request a retransmission from the AP in the case of a missing or invalid fragment, but this is beyond the scope of this paper.
3) This demonstration assumed the use of a dedicated network for the stuffed beacons, however, it would usually make more sense to use an existing network. In the case of creating a "dummy" network, one should consider how to manage down-time. For covert applications, it would be better to be consistent and continuously broadcast beacons, even when they don't contain any additional information. For other applications, it may make more sense to only broadcast beacons when there is information to transmit, thereby reducing bandwidth usage further.
4) For the sake of simplicity, we used the fragment ID present in the MAC header of the frame. However, this is treated specially by clients and analysis tools, usually being presented differently than un-fragmented packets. If the goal is to communicate discreetly, this may not be ideal. If the data is expected to be larger than a single IE, a custom fragmentation protocol can be designed, perhaps utilizing the vendor type byte.
5) The techniques discussed in this paper require access to and modification of the firmware controlling the network interfaces on both the AP and client devices. It may be possible to perform a man-in-the-middle style intercept to modify beacon packets from another AP, but that is beyond the scope of this paper.

VIII. FUTURE WORK

This paper focused on demonstrating the feasibility of this technique by using a USB Wi-Fi interface. The next step would be to implement something similar on an actual AP or router. As the interface used here was not responsible for managing real networks, the performance impact could not be measured. Performing similar tests on an active AP would reveal how many resources are required.

Additionally, this research was focused on targeting layer 2 of the OSI model. If a similar technique could maintain secrecy while utilizing layer 3, direct firmware modification may not be required. Layer 3 would also make the technique more protocol agnostic, potentially working with other communication methods such as 802.3 Ethernet and LTE data rather than relying on 802.11 Wi-Fi beacons.

REFERENCES

[1] R. Chandra, J. Padhye, L. Ravindranath, and A. Wolman, Beacon-Stuffing: Wi-Fi without Associations, *Eighth IEEE Workshop on Mobile Computing Systems and Applications*, 2007.

[2] 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, pp. 612, Feb. 2013.

[3] M. S. Gast, *802.11 Wireless Networks: The Definitive Guide*, 2nd ed. Beijing: O'Reilly, 2013.

[4] P. Biondi, Scapy, *Scapy*. [Online]. Available: https://scapy.net/. [Accessed: 02-Aug-2018].

[5] Fernet (symmetric encryption), *Welcome to pyca/cryptography*. [Online]. Available: https://cryptography.io/en/latest/fernet/. [Accessed: 02-Aug-2018].

[6] Wireshark, *Wireshark Documentation*. [Online]. Available: https://www.wireshark.org/. [Accessed: 02-Aug-2018].