Abusing the Windows WiFi native API to create a Covert Channel

Andrés Blanco, Ezequiel Gutesman
Core Security Technologies

April 13, 2011

Abstract

Communications over wireless channels have been perfectioned in the last years mainly improving performance and speed features. Security in this field has been a concern since the first 802.11 draft, having evolved by adding security features based on different crypto systems. In this paper we focus on the construction of a covert channel, exploitable in any system, between any endpoint and a specially crafted endpoint. The channel built can be started even while an active connection is established between a computer and a wireless Access Point, with one unique network device. This functionality allows an attacker that compromised a wireless enabled endpoint to extract available information and avoid detection. We will describe the design behind the channel structure and a fully functional implementation.

1 Introduction

Several factors must be considered in order to develop a functional "guest" covert channel "hosted" in any medium or protocol. First, one must decide which legitimate information units from the "host" protocol will be used to carry the covert channel messages. Second, the limitations of the covert channel such as speed or bandwidth are directly established by the first choice. This work has as its main purpose to design a covert channel over 802.11, giving connectivity to the attacker's machine from a low-privilege application running on the victim's machine.

Our purpose was to establish a communication between two peers, A and B. Assuming:

- Peer A represents the victim and could be connected to a legitimate wireless network.
- We have a userland process runnning on peer A, no privilege escalation seems to be possible. We want to establish a connection through wireless medium with this process.
- Peer B is the attacker's machine, fully controlled. The only requirements for this peer is that it must have injection capabilities, being able to inject for example, management frames (i.e., raw injection).

- There could be other peers, say A' that could also be connected through the covert channel.

The next sections describe the decisions taken in the design, a description of the abused features inside the Windows WiFi native API and how the channel could be used in a real scenario.

2 802.11 Management Frames Overview

A radical difference exists between wired and wireless networks. While trying to identify a network station or when an identified network station wants to connect with a network several steps must be followed. 802.11 management frames are used to perfom tasks that may be trivial in wired networks but that in a wireless medium require some special attention. For example, mobile stations searching for connectivity must locate compatible Access Points (AP). After identifying the network, the mobile station must be authenticated by the AP and once authenticated it must associate with it in order to get access to the network. In a wired environment some of these tasks are performed by simply throwing a cable between the station and a central node (e.g., a network switch).

2.1 Structure of Management Frames

All types of management frames (see [80205]) share the same header information in the Medium Access Control (MAC) layer but differ in their body. This structure is illustrated in figure 1.

Source address (SA) and destination address (DA) are used to define the frame's direction. Some management frames are used to maintain properties inside a Basic Service Set (BSS[1]). The BSSID field is used in order to limit the effect of broadcast and multicast. Theoretically, only broadcast and multicast frames from the BSSID that the mobile station is currently associated with should be passed to MAC management layer, the only exception for this rule are the *beacon frames*.

We will focus on the process of AP announcing & discovery. This process involves both the mobile station and the AP and use special types of managenemt frames: the *Beacon*, *Probe Request* and *Probe Response* frames.

[1]The building block of 802.11 networks. A *BSS* is a set of stations that are logically associated with each other [80205].

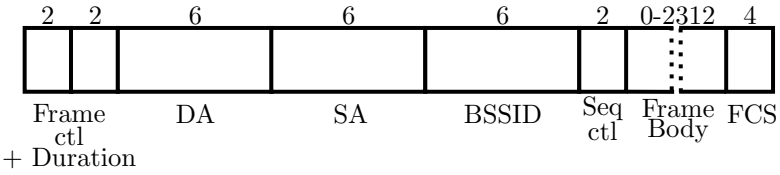

Figure 1: MAC 802.11 management frame header.

2.2 Interesting Management Frame Sub-Types

Beacon Frames This management frames announce the presence of a network. They are transmitted in regular intervals and carry information about the network (such as the security algorithm used, transmission properties, etcetera). The BSS is defined through the beacon range. As shown in figure 2 Beacons add 3 mandatory fields to the common MAC header shared by all Management frames: *Timestamp*, *Beacon Interval*, *Capability bitmap* and *SSID*.

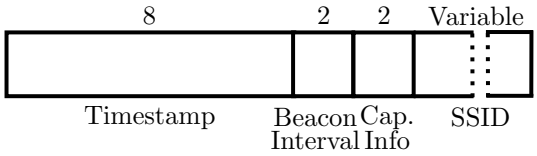

Figure 2: Beacon frame body mandatory fields.

Beacons are received from any client inside the beacon range. In MS Windows systems, some fields can be read by userland processes, giving different "Network monitors" access to the information present in the beacons, this is a crucial property that will be very useful for our purposes.

Probe Request Frames Mobile stations scanning for available (and ussually known) networks send probes. In a *Probe Request* frame the stations could specify the requested SSID and the rates supported by the station. In order for a station to be able to join a BSS, it has to support all the rates supported by the BSS and announced in the *Beacon* frames. Figure 3 shows the fields in a *Probe Request* frame.

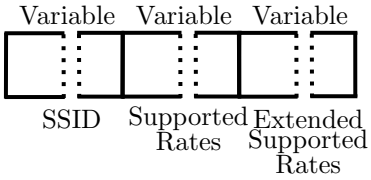

Figure 3: Probe Request frame body fields.

Probe Response Frames Once a *Probe Request* is received by a network with compatible parameters it sends a *Probe Response* frame. The responsiblity of answering a *Probe Request* lies on the station that sent the last *Beacon* frame. While in infrastructure networks this responsibility goes to the AP.

The structure of a *Probe Response* carries the same information found in a *Beacon* frame adding some specific fields which give details about the channel, such as frequency hopping properties.

3 Previous Work

The idea of a covert channel over 802.11 has been mentioned in previous works [Oud] and some of them [LB] have made different implementations (e.g. Raw covert) that needed special driver patches or worked for some OSs or drivers. Raw Covert [LB] used control *ACK* frames to encapsulate a proprietary protocol, it had some limitations on the drivers used and was designed to communicate two "hidden" peers. The present covert channel implementation uses some ideas presented by previous work but never seen implemented. It works on any Windows operating system (starting from Windows XP^{TM}service pack 2 through Windows 7^{TM}) without any specific driver. The main difference between the present work and previous prototypes is that the covert channel implemented can be established even from a compromised host that has an active 802.11 conection.

It is intended, for example, to be reliable in a post-exploitation phase[2], easy to use as an alternative communication channel with a compromised host.

According to Lampson [Lam73] a communication channel is covert "if it was neither designed nor intended to transfer information at all; it uses entities not normally viewed as data objects to transfer information from one subject to another" [Kem83].

Hiding information inside network protocols [HS96] has been broadly studied and resulted in a more specific definition for covert channels including properties such as indistinguishability and detectability. This means a covert channel cannot be distinguished among normal traffic, no matter whether it's encrypted or not.

In the context of the implementation of this covert channel we should say that if we encrypt the information contained inside the 802.11 frames, we could pass as undetectable , this has a relation with the work presented in [DZM05] that explains that Windows XP injects Probe Requests with random data inside the SSID IE. So it would be difficult to distinguish between real probes or the covert channel implementation. For the purpose of the PoC encryption features were not included.

4 Covert Channel Design

The main purpose in the construction of a wireless channel is to take advantage of some particular OS-level functionalities related to wireless connectivity by using them for a covert communication channel. The particularity of these functionalities is that they can be used without having special privileges or enabling special driver modes on the wireless cards such as the monitor mode.

[2]The tasks performed after taking control over a system, with either high or low privileges.

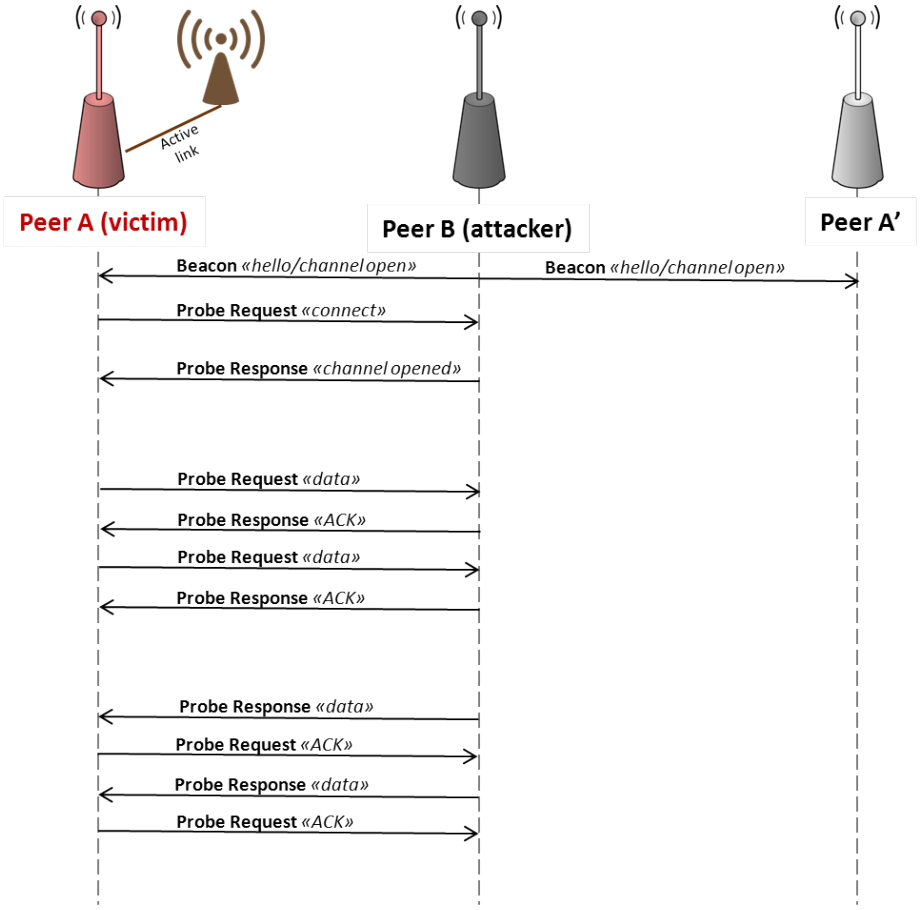

Figure 4: Channel messages between peer A (victim), peer B (attacker) and a potential victim.

Figure 4 shows one possible scenario where this channel could be used to connect two peers, A and B, where peer A has an active connection with an AP. There is a tiny protocol which controls the channel.

In order to exchange information we will use the types of management frames described in section 2.2 as our channel packages. For this purpose, the channel must be encapsulated into valid management frames.

Management frames share the same MAC layer header and depending on the frame subtype, they specify different *information elements* which carry specific information for each subtype.

Probe Request frames are the management frame subtypes used to communicate from the victim to the attacker's machine. According to the 802.11 specification [80205], a client sends *Probe Requests* when it tries to find available (previously known) networks.

Since *Probe Request* are part of the management frame type they have a *BSSID* field which is a 6-octet MAC address that identifies the AP that the client is trying to connect to. The information element carried inside *Probe Requests* is the *SSID* element, a 32-byte element used to carry the "friendly name" of the

network (e.g. the one seen while browsing networks from a connection manager while reading *Beacon* frames from the AP).

With these two fields we have a total of 38 bytes for each packet in our channel. Table 4 specifies how these bytes are used.

Mgmt. Field	Byte	Value
BSSID	$0 \sim 2$	Signature
	3	Flags
	$4 \sim 5$	Seq. No.
SSID	$0 \sim 32$	Packet Data

Table 1: 802.11 Packet fields mapping.

Section 5 further describes implementation details on the Windows WiFi native API [Win].

5 Implementation Details

As described in section 4, we need to be able to process information from *Beacon* and *Probe Response* frames and send information through *Probe Request* frames. In Microsoft Windows systems the Native Wifi API provides all the functionality needed. The key functions to create the covert channel are the following:

- `WlanScan`
- `WlanGetNetworkBssList`

The `WlanScan` function is used to inject data through *Probe Request* frames and `WlanGetNetworkBssList` to read data from *Beacon* and *Probe Response* frames. With this two functions we have a channel.

The signature of `WlanScan` follows:

```
DWORD WINAPI WlanScan(
  __in        HANDLE hClientHandle,
  __in        const GUID *pInterfaceGuid,
  __in_opt    const PDOT11_SSID pDot11Ssid,
  __in_opt    const PWLAN_RAW_DATA pIeData,
  __reserved  PVOID pReserved
);
```

As we can see above, there are two parameters that can set data inside the *Probe Request* frame. The `pDot11Ssid` parameter that sets the SSID Information Element and the `pIeData` that sets a custom Information Element.

In the case of `pDot11Ssid` parameter we have a pointer to a `DOT11_SSID` structure:

```
typedef struct _DOT11_SSID {
  ULONG uSSIDLength;
  UCHAR ucSSID[DOT11_SSID_MAX_LENGTH];
} DOT11_SSID, *PDOT11_SSID;
```

This structure has two members:

- `uSSIDLength`: The length, in bytes, of the `ucSSID` array.
- `ucSSID`: The SSID. `DOT11_SSID_MAX_LENGTH` is set to 32.

This imposes the 32 bytes data limit.

In the case of `pIeData` parameter we have a pointer to a `WLAN_RAW_DATA` structure:

```
typedef struct _WLAN_RAW_DATA {
  DWORD dwDataSize;
  BYTE  DataBlob[1];
} WLAN_RAW_DATA, *PWLAN_RAW_DATA;
```

This structure has two members:

- `dwDataSize`: The size, in bytes, of the DataBlob member. The maximum value of the dwDataSize may be restricted by the type of data stored in the `WLAN_RAW_DATA` structure.
- `DataBlob`: The data blob.

In this case we have a limitation of 240 bytes that is defined by the `DOT11_PSD_IE_MAX_DATA_SIZE` constant. As a side note we should notice that injecting data on an Information Element works only in drivers that support it.

The `WlanGetNetworkBssList` function is going to return the data acquired through *Beacon* and *Probe Response* frames. This can be seen as the *INPUT* side of the channel. It has the following parameters:

```
DWORD WINAPI WlanGetNetworkBssList(
  __in        HANDLE hClientHandle,
  __in        const GUID *pInterfaceGuid,
  __opt       const  PDOT11_SSID pDot11Ssid,
  __in        DOT11_BSS_TYPE dot11BssType,
  __in        BOOL bSecurityEnabled,
  __reserved  PVOID pReserved,
  __out       PWLAN_BSS_LIST *ppWlanBssList
);
```

For our purposes, the most important parameter of this function is the last one, the pointer to a `WLAN_BSS_LIST` structure:

```
typedef struct _WLAN_BSS_LIST {
  DWORD          dwTotalSize;
  DWORD          dwNumberOfItems;
  WLAN_BSS_ENTRY wlanBssEntries[1];
} WLAN_BSS_LIST, *PWLAN_BSS_LIST;
```

[2]The `pIeData` only works with some drivers. A workaround could be that the protocol could test if the driver supports the injection of the `pIeData` and in the case it works set the application to use this extra data setting a bigger MTU.

This structure is a list of WLAN_BSS_ENTRY structures each one containing the information of the BSS[3]:

```
typedef struct _WLAN_BSS_ENTRY {
  DOT11_SSID        dot11Ssid;
  ULONG             uPhyId;
  DOT11_MAC_ADDRESS dot11Bssid;
  DOT11_BSS_TYPE    dot11BssType;
  DOT11_PHY_TYPE    dot11BssPhyType;
  LONG              lRssi;
  ULONG             uLinkQuality;
  BOOLEAN           bInRegDomain;
  USHORT            usBeaconPeriod;
  ULONGLONG         ullTimestamp;
  ULONGLONG         ullHostTimestamp;
  USHORT            usCapabilityInformation;
  ULONG             ulChCenterFrequency;
  WLAN_RATE_SET     wlanRateSet;
  ULONG             ulIeOffset;
  ULONG             ulIeSize;
} WLAN_BSS_ENTRY, *PWLAN_BSS_ENTRY;
```

Since this structure provides a lot of information about the BSS it could be used as input data for the channel.

6 Attack Vectors

Technological complexity in daily-used devices turned the traditional perspective of the network boundaries obsolete [Ric08]. The scenario where an attacker penetrates an infrastructure from the "only" connection that joins the outer world with, for example, an enterprise network cannot be taken seriously.

Client side attacks, web application attack vectors and connectivity provided by mobile devices and portable computers, among others, opened new ways of compromising the networks, presenting new challenges to the ones in charge of protecting them.

The presented work can be used as a way out (or way in, depending on which side you are standing) from the classical perimeter notion. An already compromised computer, which could have been an employee's notebook attacked while surfing the web in a coffee shop can be used as a gate to a given private network. It could be running a "channel deamon" listening for available wireless networks that when receiving a special crafted packet opens the connection through the covert channel without the users notice.

In penetration testing terms, compromised machines (e.g., running network agents or advanced payloads [Cac02, Met]) could loose connectivity and might use wireless covert channels to try to establish a connection back to the penetration tester console, bypassing all the security infrastructure the network could have (e.g., firewalls and traffic rules). Figure 5 shows a possible scenario where a wireless covert channel could be useful.

[3] basic service set.

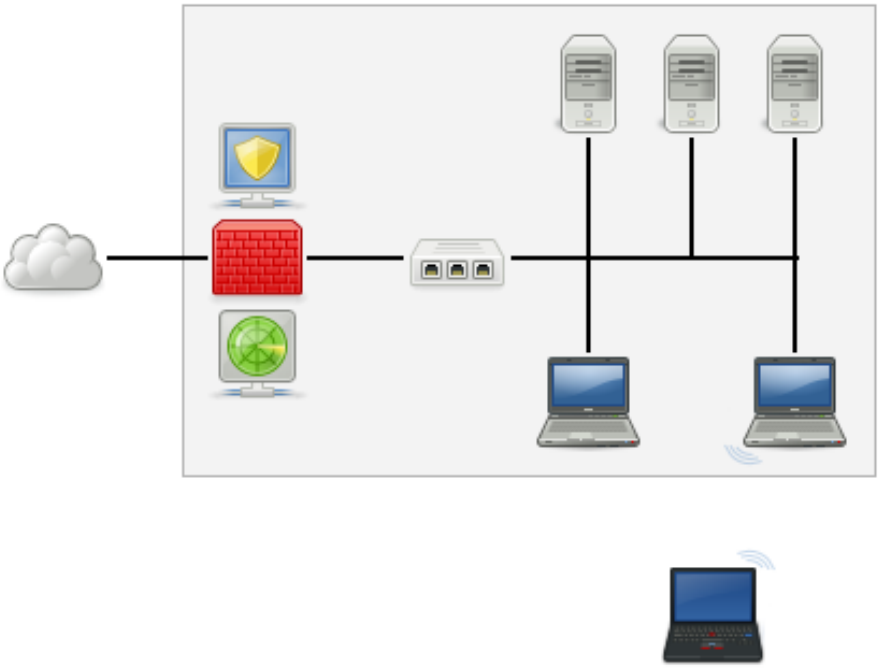

Figure 5: Sample network boundaries scenario for covert channel attacks.

Another example could be compromising a machine with a client-side exploit using a pen drive as the attack vector and connecting back from the compromised target. In this scenario, if the machine lacks a network connection, the attack will fail. Using the covert channel could allow communication with the target without any connection besides the wireless covert channel.

Figure 6: Sample of an offline attack using the covert channel.

7 Conclusions

Limitations exist both on the *INPUT* and *OUTPUT* data we can use for the channel which have a direct impact on its bandwidth.

The most significative advantage of using the WiFi Native API is that no special privileges are required and the state of the Wireless Network Interface is not a limitation, since it can be joined to a wireless network or not. Morover, it works with any Wireless Network Interface that works in MS Windows with no special drivers.

Another important advantage for using this API is that the end user never notices that someone is using the Wireless Network Interface, and the only "sign" of the channel existance are the network names injected in the beacons that announce the existence of the channel, which have an empty value. Once the channel is open, directed *probe responses* and *probe requests* are used to communicate, so a "normal" user (i.e., anyone not looking for a covert channel) would not detect it.

With this advantages we could say that implementations of this covert channel are a good options besides similar attacks usign SoftAP and that are a real threat for network security.

8 Future Work

There are various features to be implemented in the prototype code. First, multi client support has to be added in order to be able to communicate different clients with one server (attacker).

Cryptographic armoring can be also implemented although the small bandwith of the channel should be analyzed in order to decide which cryptographic scheme to adopt.

Another operating system clients (the piece of code running on the attacked system) can also be implemented but still need deeper research. We currently focused in MS Windows systems in order to take advantage of the Native WiFi API provided in such systems.

References

[80205] *802.11 Wireless Networks, The Definitive Guide.* O' Reilly, 2005.

[Cac02] Maximiliano Caceres. Syscall proxying - simulating remote execution. Corelabs Technical Report, 2002.

[DZM05] D.A. Dai Zovi and S.A. Macaulay. Information assurance workshop. proceedings from the sixth annual ieee smc, 2005.

[HS96] T. G. Handel and M. T. Sandford. *Lecture Notes in Computer Science*, chapter Hiding dat. Springer Berlin / Heidelberg, 1996.

[Kem83] Richard A. Kemmerer. Shared resource matrix methodology: an approach to identifying storage and timing channels. *ACM Trans. Comput. Syst.*, 1(3):256–277, 1983.

[Lam73] Butler W. Lampson. A note on the confinement problem. *Commun. ACM*, 16(10):613–615, 1973.

[LB] Franck Veysset Laurent Butti. Wi-fi advanced stealth. In *Black Hat Briefings USA 2006*.

[Met] Meterpreter payload. URL: http://www.metasploit.com/.

[Oud] Laurent Oudot. Wlan and stealth issues. In *Black Hat Briefings Europe 2005*.

[Ric08] Gerardo Richarte. Evolution of penetration testing, 2005 to 2013. SANS What Works in Penetration Testing & Ethical Hacking Summit. Las Vegas, NV, May 31 - June 9, 2008., 2008.

[Win] Windows wifi native api reference. URL: http://msdn.microsoft.com/en-us/library/ms706275(v=VS.85).aspx.