Wireless security isn't dead, attacking clients with MSF

Abstract

Security has been an issue on wireless networks since their inception, gradually moving from insecure open networks to insecure WEP networks to more secure WPA and WPA2 networks. At this point, enterprise-level wireless network security is well understood, however client security in an environment where users are free to take systems from a secure network to an insecure network remains a significant and under-explored risk.

Bio

Mike Kershaw / Dragorn is the author of Kismet, a multi-protocol (but predominately 802.11) packet sniffer, wardriver, and IDS system, as well as LORCON, a packet injection framework for 802.11, and various and sundry additional OSS tools, usually wireless related.

Overview of 802.11

802.11 networks are defined by several basic packet types:

1. Data frames, which contain the actual data on the network. Data frames can be encrypted at layer2 (WEP, TKIP, or AES) on encrypted networks, and can be easily translated between 802.3 Ethernet and 802.11 formats. Placing an 802.11 device in promiscuous mode will usually show the data frames for the receiving station in 802.3 format, however it is not possible to show the other packet types, and it is not typically possible to receive data frames from other stations.

Data frames contain no innate authentication unless they are encrypted with a strong standard (ie TKIP or AES). Open and WEP networks contain no authentication or replay protection.

2. Management frames, which define the network and control operations. Beacon frames contain the network name (SSID), cryptographic settings, power save controls, etc. Probe request and response frames control client access to the network at a basic level.

Management frames contain no authentication mechanisms, however standards such as 802.11w and custom vendor add-ons seek to address this lack. Both require support both in the AP and in the client.

3. Control frames, which handle clear-to-send/request-to-send, data recipt acknowledgement, and power save.

802.11 networks are fundamentally shared media, as all packets are sent on the same channel and are viewable by all. Per-user encryption options such as WPA and WPA2 can introduce a "virtually switched" layer, where non-broadcast packets are decryptable only by a recipient with the per-user encryption key.

Unlike 802.3 networks which have a source MAC and destination MAC address in each frame, 802.11 data frames contain three MAC addresses (or sometimes four, in the case of mesh distribution systems):

1. The source MAC address. This is the original source, which may be a wired MAC in the case of a bridged wired/wireless network, the MAC of the wireless client, or the MAC of the access point if it is also acting as a router.

2. The destination MAC address. This can, like the source, be a wired device, the MAC of the AP if it is a router, or the MAC of another wireless device.

3. The BSSID (Basic Service Set Identifier) MAC address. This is the MAC address of the access point which is receiving and handling this packet. In a multi-access point network, this is the access point the user is currently associated with.

Monitoring 802.11 requires a device and driver which is capable of entering "monitor" (or "rfmon") mode. Nearly all hardware is capable of this mode (barring some specific embedded devices with limited firmware). Driver ability varies; Most drivers on Linux are capable of monitor mode, as are the Airport drivers on OSX, and many drivers on *BSD. Most Windows platform drivers are not, however this may change in the Vista and Windows 7 models. CACE Technologies produces a device, AirPCAP, specifically designed for monitor mode capture on Windows.

While normally an 802.11 device presents as an 802.3 Ethernet interface, in monitor mode it switches to a full 802.11 DLT (Data Link Type) and reports all decoded packets. These include all data frames, management frames, and control frames seen, for all networks on that (and adjacent, due to overlap) channels.

Current 802.11 Defenses

When we say that 802.11 access point security is well understood at this point, what we are basically saying is that we have good cryptography and that properly configured users on a WPA-Enterprise access point can trust that they are connected to a legitimate access point.

WPA-Enterprise (WPA and WPA2 using EAP authentication methods) layers additional defenses over the 802.11 layer. Per-user authentication (typically against a radius backend) is protected by x509 certificates, user traffic is segregated by per-user keys, and both TKIP and AES provide replay protection on data frames.

While this model has opportunities for failure (such as improperly configured clients accepting invalid certificates, untrained users accepting invalid certificates, vulnerabilities discovered in TKIP replay protection), these failure points can often be avoided and are not truly within the scope of this effort.

Access point centric defense mechanisims on less-secure networks typically include passive monitoring, such as detecting unauthorized access points advertising the same SSID, and active counter-attacks, such as forcing clients previously to disconnect from an unauthorized access point by spoofing deauthentication and disassociation packets.

Access point client protection is generally controlled by filtering packets at layer two in the access point itself, preventing wireless clients from communicating with each other. On most platforms this is known as “inter-client protection”. Some vendor platforms may provide additional security mechanisms, such as stateful filtering at layer two or specific service filtering between clients.

There are few defenses against denial of service attacks (barring the “crowbar” method, in other words, find the offender, and hit them with a crowbar) as the 802.11 management frames are not protected. Even an otherwise secure WPA-Enterprise network will fall to a flood of disassociate packets, and no network can withstand a broad-spectrum jammer flooding the channel with noise.

802.11 Injection and Spoofing Attacks

802.11 networks are vulnerable to several injection and spoofing attacks.

The most basic spoofing attack is to bring up a hostile access point with the same SSID as the target. The 802.11 protocol considers all access points with the same SSID to be part of the same network, and will roam to the access point with the strongest signal. This attack is the most obvious, but still highly effective.

A more subtle attack, and the one which we are most interested in, is injecting data frames directly to the client. By spoofing the BSSID of the access point, it is possible to bypass any protection the access point may offer; The packet is sent directly to the target, and cannot be filtered by the access point. This attack is trivial against open, unencrypted networks, and only slightly more difficult when targetting WEP networks, as the same key is used for all users on the network and is trivially cracked. A direct-inject attack is theoretically possible against WPA-PSK networks where the PSK is known and a client is observed associating to the network – when the PSK is known and the association handshake is observed, the PTK per-user key can be extracted and the traffic decrypted. There are further implications for implementing this attack against WPA-PSK (such as not disrupting the data sequence number protection in TKIP and AES) which have not been explored.

LORCON

LORCON (or Loss Of Radio CONtrol, http://802.11ninja.net) is a packet injection library for trivializing the per-driver problems of injecting raw 802.11 frames. Started in 2004 by Josh Wright and Mike Kershaw, the latest release (Lorcon2-2009-11) introduces libpcap integration and a pcap-like API. Developers familiar with the libpcap API will find implementing 802.11 packet handling trivial.

Currently the LORCON2 release targets the Linux kernel driver layer, mac80211. Support for older drivers such as madwifi and non-Linux drivers such as AirPCAP will be restored soon.

A typical LORCON2 program would look like:

```
…
/* Driver reference */
lorcon_driver_t *dri;
/* Context used for tracking lorcon state */
lorcon_t *ctx;
/* Packet data, initialized however */
uint8_t packet[...];

/* Automatically determine the driver */
dri = lorcon_auto_driver(“wlan0”);

/* Create the context */
ctx = lorcon_create(“wlan0”, dri);

/* Open in monitor + inject mode */
lorcon_open_injmon(ctx);

/* Set channel */
lorcon_set_channel(ctx, 6);

/* Send arbitrary bytes */
lorcon_send_bytes(ctx, sizeof(packet), packet);
```

. . .

LORCON2 allows application developers to abstract away from specific driver limitations and problems.

Attacking The Clients

Currently under-represented in the wireless security space is the risk to clients. Clients can be secured while connected to their “base” network (for example, a strongly secured WPA2-Enterprise network at the corporate office), however wireless systems are (typically) also mobile systems, and are then placed in the hands of users who take them home, to the bookstore, to the coffee shop, airports, etc. Several risks remain even for users trained (or mandated) to use a VPN or similar tecnology.

Services which retain state across security boundaries are vulnerable to attacks from open networks. As shown by Rsnake / Robert Hanson in “RFC 1918” blues” (http://www.sectheory.com/rfc1918-security-issues.htm), a significant exposure exists in the browser caching mechanism.

If an attacker is able to control both the remote host and the cache control of a web page, a malicious page may be cached by the browser across security domains. Once cached, the page is loaded from the local cache, even when the user is no longer within the control of the attacker.

To implement this on 802.11, we look to a much older tool. Debuted at Defcon many years prior by Toast, AirPwn implemented TCP session hijacking over wireless. By updating this tool and integrating with Metasploit, we leverage the fledxibility of MSF and LORCON2.

TCP Hijacking

The basic methodolgy used for TCP hijacking is identical to historic attacks against TCP sessions. A TCP session is only “secure” against attack in as much as the sequence and acknowledgement numbers are unknown to an attacker. On an open, shared media network this is no longer the case. The TCP session hijack is implemented as a man-in-the-middle race condition, relying on the fact that the remote HTTP server is likely to be further away compared to the attacker.

A typical HTTP session:

Client → Server “GET /foo.html HTTP/1.0”
sequence 123 ack 456

Server ← Client "Content-Length: 123 Content-Type: Text/HTML <html>..."
sequence 456 ack 145

A HTTP session as seen by Airpwn-MSF:

Client → Server "GET /foo.html HTTP/1.0"
sequence 123 ack 456

Airpwn (spoofing server) ← Client "Content-Length: 200 Content-Type: Text/HTML Malicious page content..."
Sequence 456 ack 145

Airpwn (spoofing client) → Server "FIN!"

Airpwn (spoofing server) ← Client "Fin!"

Server (Ignored out-of-sequence data) ← Client "Content-Length: 123 Content-Type: Text/HTML <html>..."

sequence 456 ack 145

Impacts of TCP Stream Hijacking

TCP stream hijacking allows an attacker within WiFi range of the victim to replace any unencrypted content stream in such a way that the victim is unaware. Focusing on the impacts to web applications:

1. Cache control. TCP session attacks allow control of the HTTP headers, which define the cache control options. Malcious content may be stored in the victims cache for exploitation in the future.

2. Direct exploitation of browser vulnerabilities. By inserting malicious content in otherwise "trusted" or "benign" websites via TCP session hijacking, direct browser vulnerability exploitation becomes possible.

3. Complete control of the DOM. Once malcious Javascript is inserted into a page, complte control of the page is given to the attacker. Example DOM rewrite attacks include:

1. Rewriting of links to remove encryption (rewrite HTTP and POST links for HTTP instead of HTTPS), allowing the attacker to sniff login information.

2. Rewriting of form submission targets to a logging stager, allowing the attacker similar access to login details.

3. Export of cookie data to an external logger, allowing the attacker access to user sessions.

4. Dynamic replacement of website content, allowing an attacker opportunities to manipulate news items, stock reports, bank statements, or other content as viewed by the victim.

Cache Control Attacks

The TCP session attacks against the HTTP streams allow for control of the HTTP headers, which define the cache control options: The HTTP response options "Cache-control" and "Expires" control the browser retention.

By controlling the cache of the victim browser, malicious content may be stored for future use. Manipulating the cache of pages considered by the user as "benign" or "insecure" allows an attacker future access to the browser within the secure perimiter.

Poisoning using a Javascript "staging" page allows complex control in future operations: Specifically, using the cached Javascript to dynamically load the true, foreign content of the target page, then rewriting it before replacing the currently viewed document with the "legitimate" content.

Combining the staging technique with a remote site controlled by the attacker allows future attacks to be dynamically inserted into pages viewed by the user for as long as the staging page remains cached: For example, unpatched browser attacks may be leveraged against clients within a "secure" perimiter once they have been cache-poisoned via an insecure network:

1. User takes laptop from a secure location to an insecure location (corporate network to coffee shop WiFi)

2. User browses to an untrusted site (Social networking, photo sharing, news). Site does not need to require credentials.

3. MSF-Airpwn used to poison victim with cached "stager" page. User sees original content and nothing is changed. Content is actually loaded dynamically by the cached Javascript. Stager code checks attacker website for updates each time target page is loaded.

4. User returns to secure location

5. Unpatched browser vulnerability is exposed. Attacker places attack code on publicly accessible website coded into stager.

6. The next time the user views the same site, attack code is executed, allowing an attacker a path into the secure network. Attack code can be sent via HTTPS to evade detection by secure network IDS systems (should detection for the vulnerability exist).

Advanced Cache Prediction Attacks

The attacks as discussed so far require the victim to visit a site from an insecure network without a VPN. While this is likely to be the majority of users, attack avenues exist:

1. User takes laptop from secure location to insecure location.

2. User visits hotspot landing / EULA / sign-in page.

3. Attacker poisons hotspot landing page via TCP session hijacking.

4. Poisoned hotspot page loads "likely" sites in the background while the user is signing in.

5. Attacker poisons sites loaded by the hotspot. Advanced code may be used in the hotspot to detect if the TCP attack landed, and to re-request the page until the attack race condition is met.

6. Victim switches to VPN.

7. Victim visits page poisoned by the hotspot stager code.

8. Victim has cached stager code for the site due to opportunistic guessing.

Special Thanks

Special thanks to everyone who has helped, listen to rants, and contributed ideas, especially HD Moore, Rsnake, Toast, Jesse Burns, and Renderman.