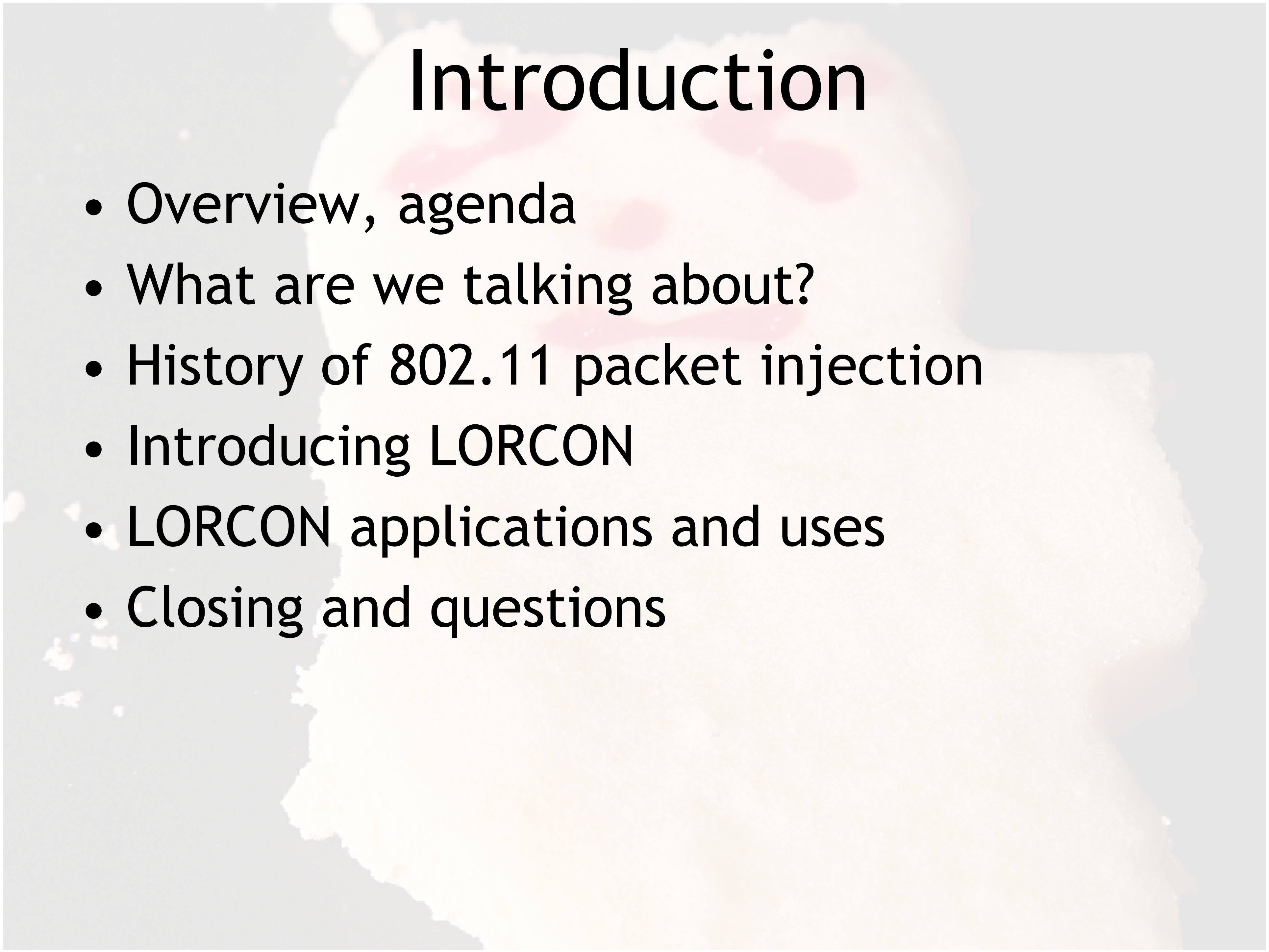
Extensible 802.11 Packet Flinging

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

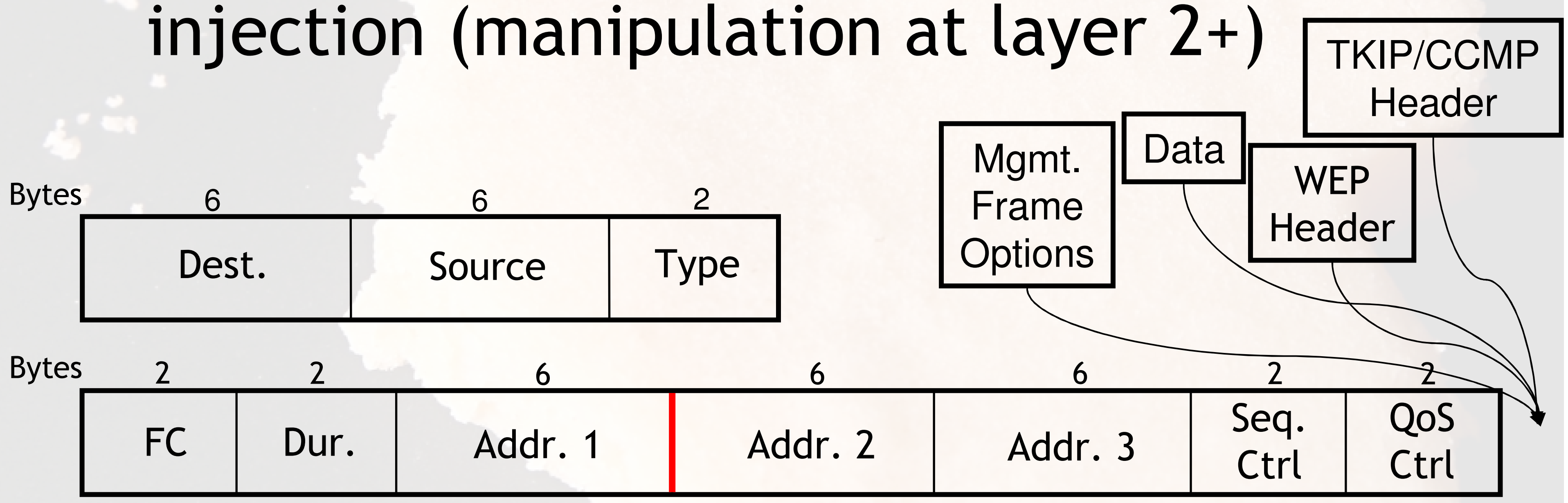
- Joshua Wright, Aruba Networks
  - Author of a bunch of one-off tools (Asleap, coWPAtty, file2air, bluepinning, jrockets)
  - Wireless security research for Aruba
- Mike Kershaw, Aruba Networks
  - Wrote Kismet, WiSpy for Linux
  - Aruba product security analyst

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+)



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 suppored by LORCON or not. Allows your app to support new cards as they are added to LORCON.	
INJ_NODRIVER	Driver name not recognized
>0	LORCON supported driver idenitfier
in_str	User-supplied driver description

```
#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

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

Initializes the handler functions, capabilities	
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()

```
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

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).	
Return >= 0	Bitmask of capabilities
Return 0	No driver capabilities found
in_tx	LORCON per-interface context

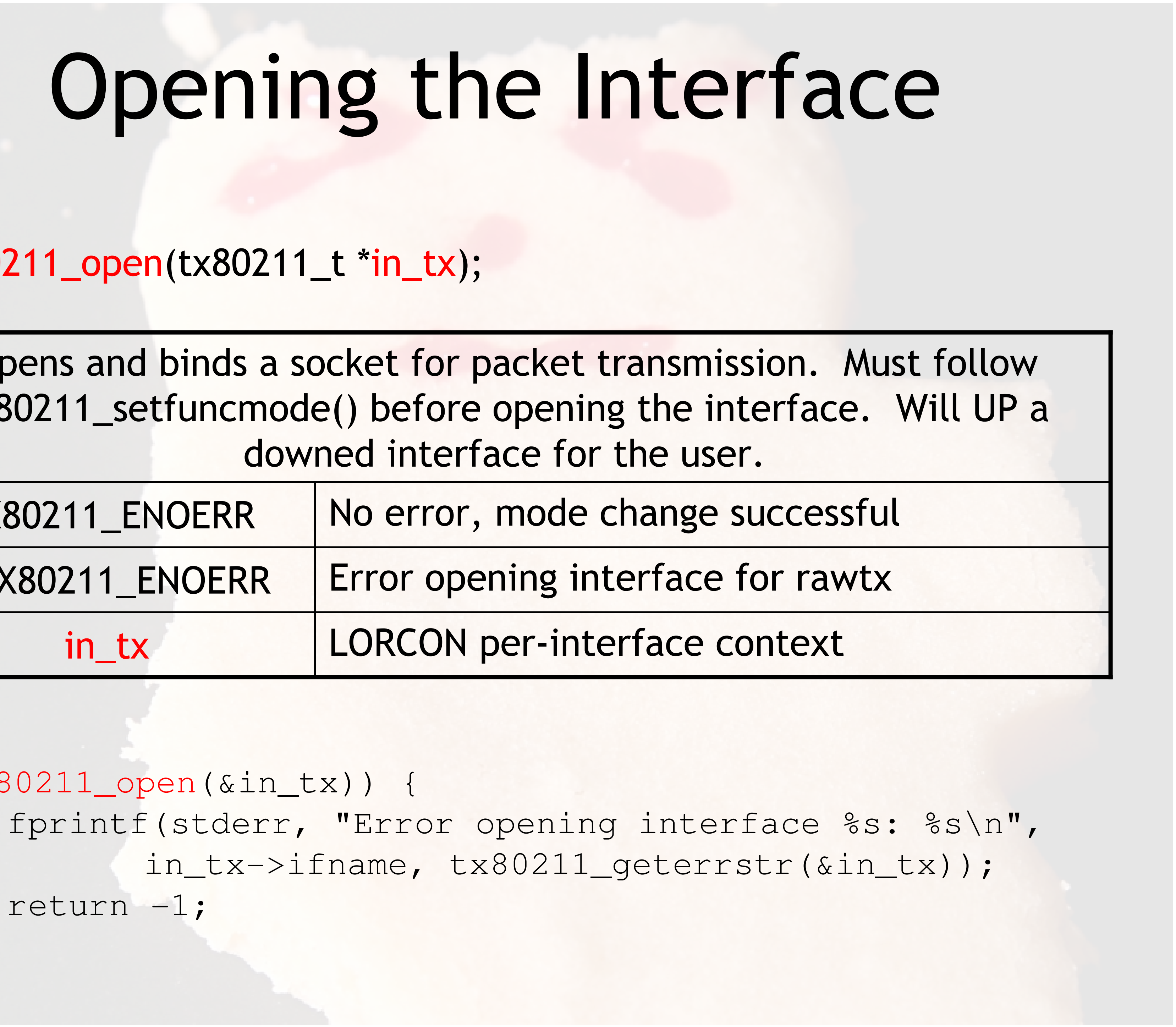
```
/* TX80211_CAP_FRAG indicates the initialized driver allows
   us to preserve the MOREFRAG bit and the fragment number
   field */
if ((tx80211_getcapabilities(&in_tx) & TX80211_CAP_FRAG) == 0)
       fprintf(stderr, "Sorry, driver does not support "
                         "manipulating fragmentation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

Operating Modes

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().	
TX80211_ENOERR	No error, mode change successful
< TX80211_ENOERR	Error setting functional mode
in_tx	LORCON per-interface context
in_fmode	Functional mode: TX80211_FUNCMODE_RFMON TX80211_FUNCMODE_INJECT TX80211_FUNCMODE_INJMON

```
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);

Opens and binds a socket for packet transmission. Must follow tx80211_setfuncmode() before opening the interface. Will UP a downed interface for the user.	
TX80211_ENOERR	No error, mode change successful
< TX80211_ENOERR	Error opening interface for rawtx
in_tx	LORCON per-interface context

```
if (tx80211_open(&in_tx)) {
        fprintf(stderr, "Error opening interface %s: %s\n",
                in_tx->ifname, tx80211_geterrstr(&in_tx));
        return -1;
}
```

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(). tx80211_packet_t is independent of the per-interface context tx80211_t to accommodate rapidly transmitting different packets.	
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);
```

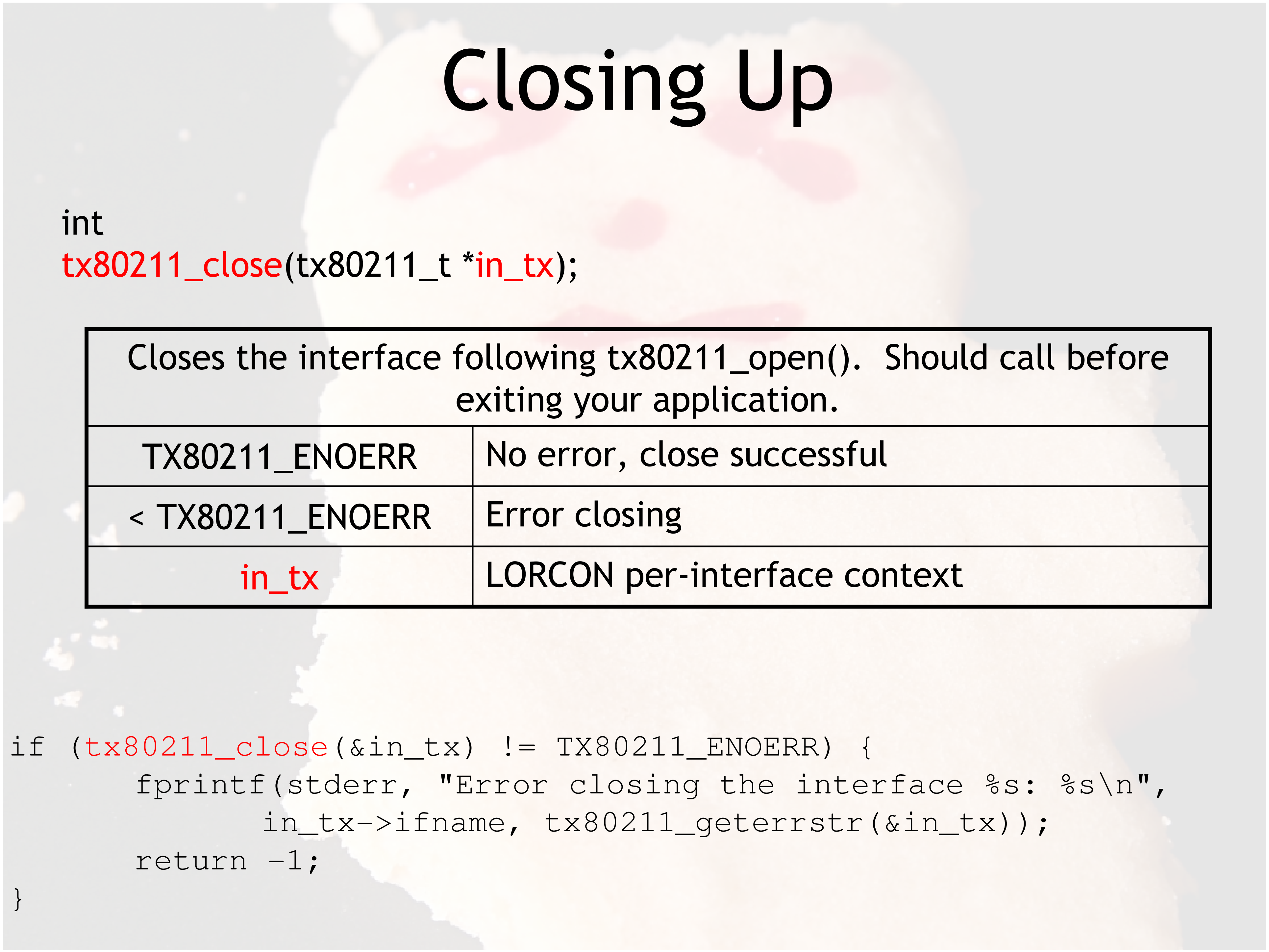
Transmitting Packets

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

Transmits the contents at in_packet->packet for in_packet->plen bytes.	
Return >0	Number of bytes transmitted
TX80211_ENOTX	0 bytes transmitted
TX80211_EPARTTX	Partial frame transmitted
in_tx	LORCON per-interface context
in_packet	LORCON per-packet context

```
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



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

Closes the interface following tx80211_open(). Should call before exiting your application.	
TX80211_ENOERR	No error, close successful
< TX80211_ENOERR	Error closing
in_tx	LORCON per-interface context

```
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;
}
```

Simple LORCON Application

```
#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

```
struct 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 it's 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?

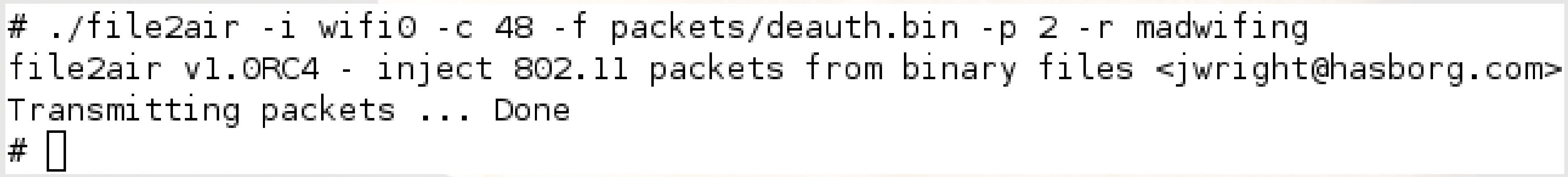
```
lcpa_metapack_t *metapack;
tx80211_packet_t txpack;
uint8_t txmac[6];
uint8_t targetmac[] =
"\xff\xff\xff\xff\xff\xff";

metapack = lcpa_init();
tx80211_initpacket(&txpack);

srand(time(NULL));
lcpf_randmac(txmac, 1);
```

```
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

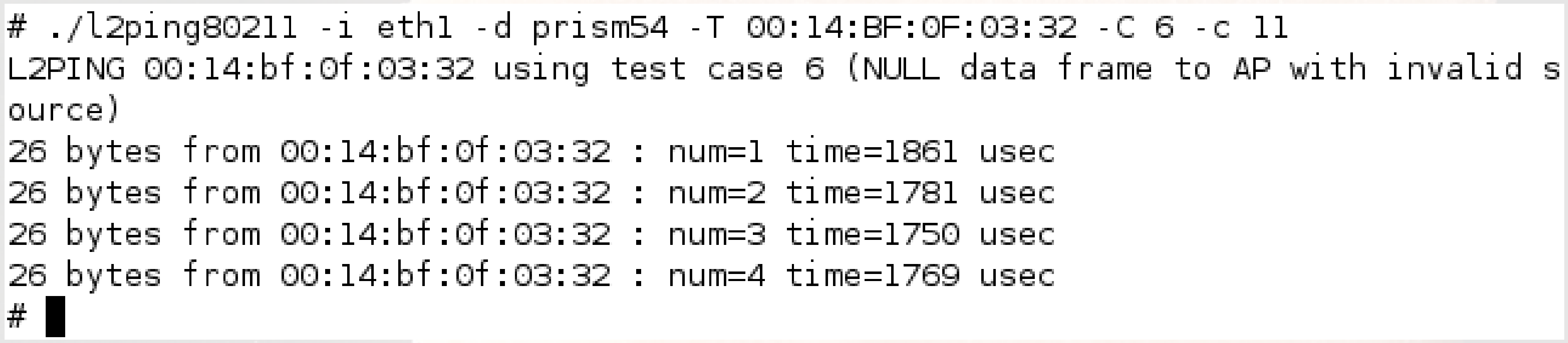


```
# ./file2air -i wifi0 -c 48 -f packets/deauth.bin -p 2 -r madwifing
file2air v1.0RC4 - inject 802.11 packets from binary files <jwright@hasborg.com>
Transmitting packets ... Done
# 
```

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

l2ping80211

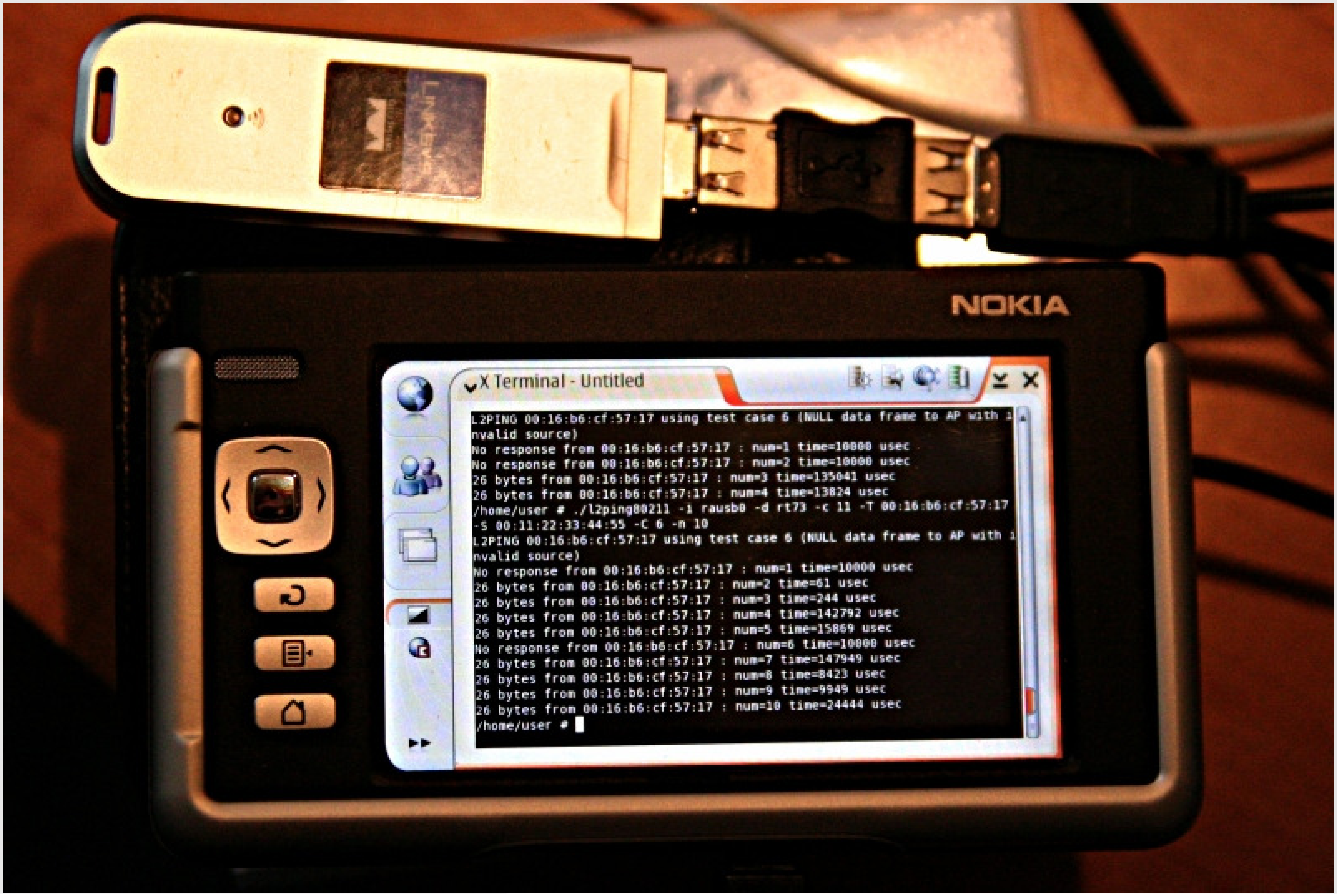
- Verifies reachability of target wireless station using various L2 tests
  - Regardless of encryption in use
- I can't think of a use for this, why would we need to repeatedly check the responsiveness of a target host?



```
# ./l2ping80211 -i eth1 -d prism54 -T 00:14:BF:0F:03:32 -C 6 -c 11
L2PING 00:14:bf:0f:03:32 using test case 6 (NULL data frame to AP with invalid s
ource)
26 bytes from 00:14:bf:0f:03:32 : num=1 time=1861 usec
26 bytes from 00:14:bf:0f:03:32 : num=2 time=1781 usec
26 bytes from 00:14:bf:0f:03:32 : num=3 time=1750 usec
26 bytes from 00:14:bf:0f:03:32 : num=4 time=1769 usec
# 
```

Sample application included with LORCON; "make l2ping80211"

LORCON on the Nokia 770



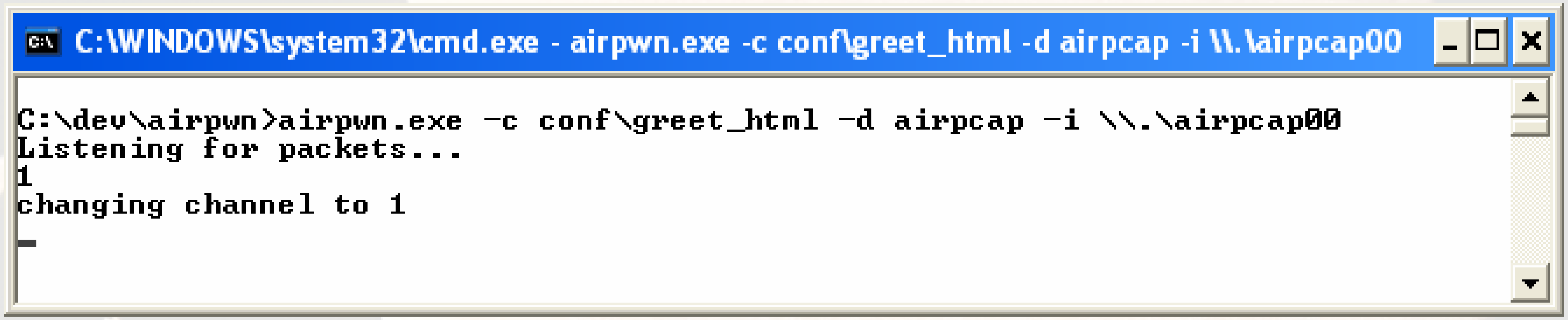
```
L2PING 00:16:b6:cf:57:17 using test case 6 (NULL data frame to AP with i
nvalid source)
No response from 00:16:b6:cf:57:17 : num=1 time=10000 usec
No response from 00:16:b6:cf:57:17 : num=2 time=10000 usec
26 bytes from 00:16:b6:cf:57:17 : num=3 time=135041 usec
26 bytes from 00:16:b6:cf:57:17 : num=4 time=13824 usec
/home/user # ./l2ping80211 -i rausb0 -d rt73 -c 11 -T 00:16:b6:cf:57:17
-S 00:11:22:33:44:55 -C 6 -n 10
L2PING 00:16:b6:cf:57:17 using test case 6 (NULL data frame to AP with i
nvalid source)
No response from 00:16:b6:cf:57:17 : num=1 time=10000 usec
26 bytes from 00:16:b6:cf:57:17 : num=2 time=61 usec
26 bytes from 00:16:b6:cf:57:17 : num=3 time=244 usec
26 bytes from 00:16:b6:cf:57:17 : num=4 time=142792 usec
26 bytes from 00:16:b6:cf:57:17 : num=5 time=15869 usec
No response from 00:16:b6:cf:57:17 : num=6 time=10000 usec
26 bytes from 00:16:b6:cf:57:17 : num=7 time=147949 usec
26 bytes from 00:16:b6:cf:57:17 : num=8 time=8423 usec
26 bytes from 00:16:b6:cf:57:17 : num=9 time=9949 usec
26 bytes from 00:16:b6:cf:57:17 : num=10 time=24444 usec
/home/user #
```

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

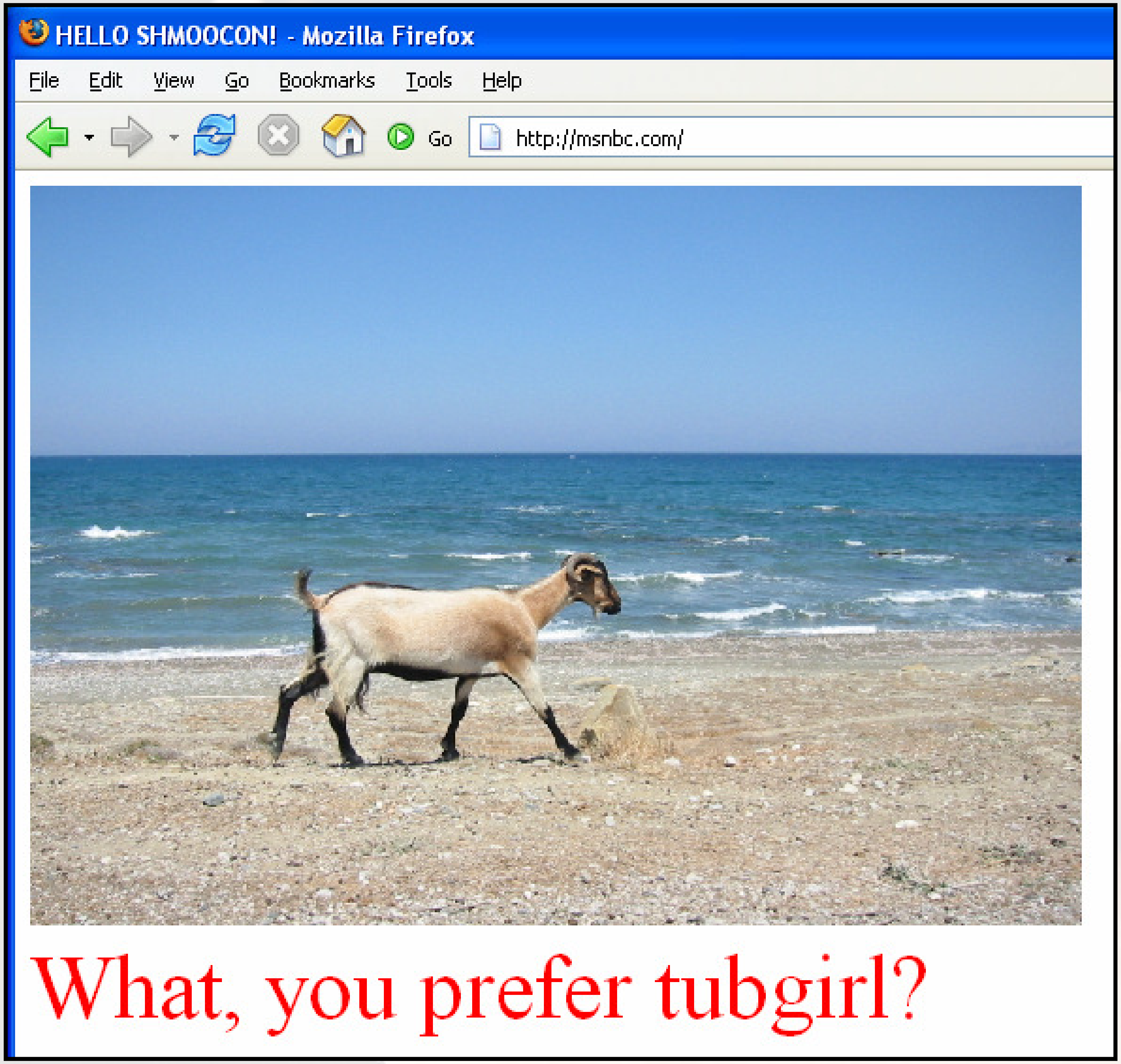
- Airpcap: Commercial adapter
  - TX support in driver beta, to be released "real soon now"


C:\WINDOWS\system32\cmd.exe - airpwn.exe -c conf\greet_html -d airpcap -i \\.\airpcap00

```
C:\dev\airpwn>airpwn.exe -c conf\greet_html -d airpcap -i \\.\airpcap00
Listening for packets...
1
changing channel to 1
_
```

AND NOW FOR A LITTLE GOATSEA

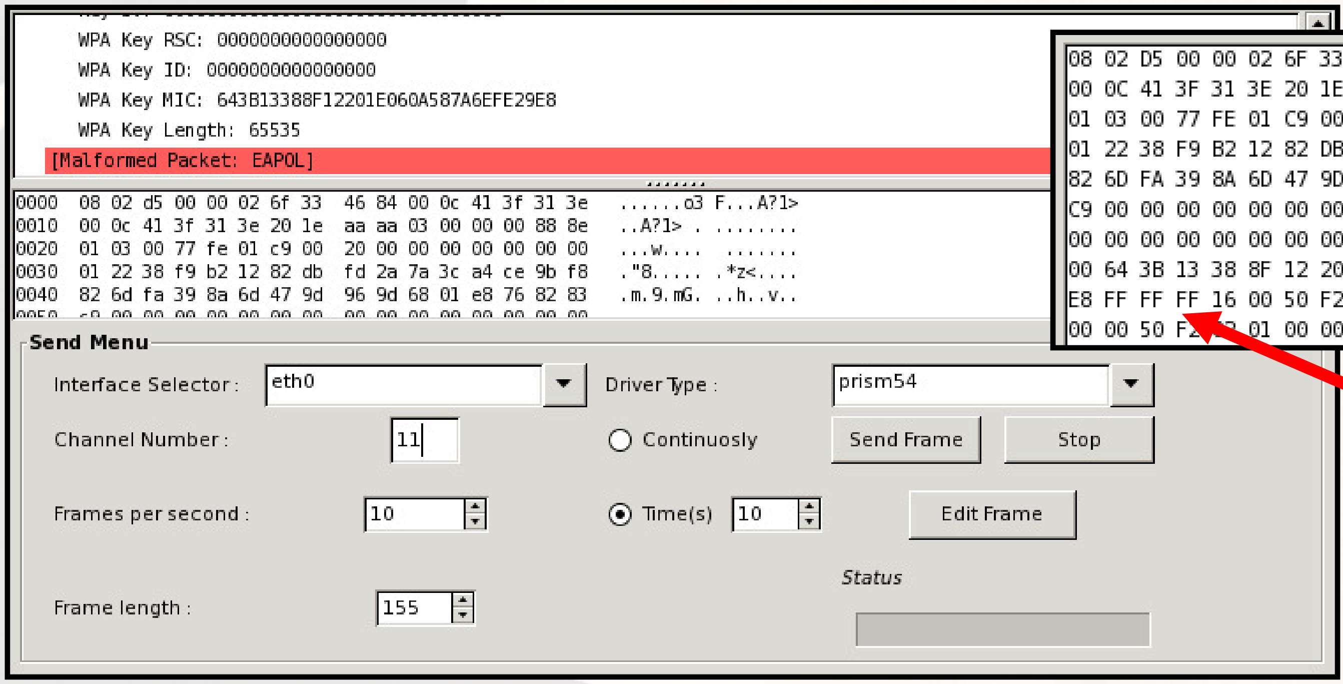
http://802.11ninja.net/code/airpwn-windows.zip


HELLO SHMOOCON! - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
Go
http://msnbc.com/

What, you prefer tubgirl?

Wireshark WiFi Injection Patch

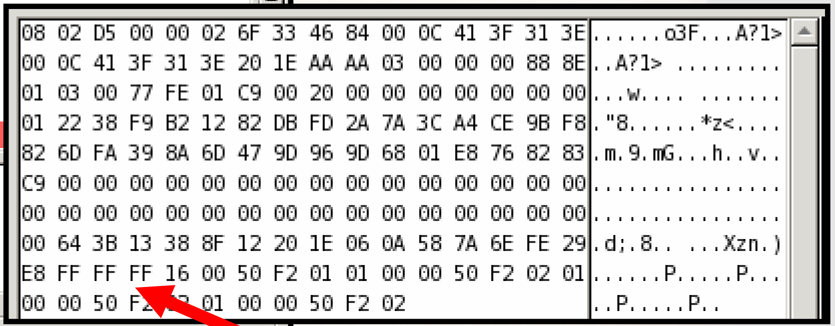
- Patch to Wireshark by Asier Martínez
  - Select 802.11 frame, r-click Packet → Send WiFi Frame
  - Use hex editor to modify, send repeatedly



```
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: 643B13388F12201E060A587A6EFE29E8
    WPA Key Length: 65535
  [Malformed Packet: EAPOL]

0000  08 02 d5 00 00 02 6f 33  46 84 00 0c 41 3f 31 3e   ......o3 F...A?1>
0010  00 0c 41 3f 31 3e 20 1e  aa aa 03 00 00 00 88 8e   ..A?1> . ........
0020  01 03 00 77 fe 01 c9 00  20 00 00 00 00 00 00 00   ...w....  .......
0030  01 22 38 f9 b2 12 82 db  fd 2a 7a 3c a4 ce 9b f8   ."8..... .*z<....
0040  82 6d fa 39 8a 6d 47 9d  96 9d 68 01 e8 76 82 83   .m.9.mG. ..h..v..
```

Send Menu

Interface Selector: eth0
Driver Type: prism54
Channel Number: 11
[ ] Continuosly
[✓ Time(s)] 10
Frames per second: 10
Frame length: 155



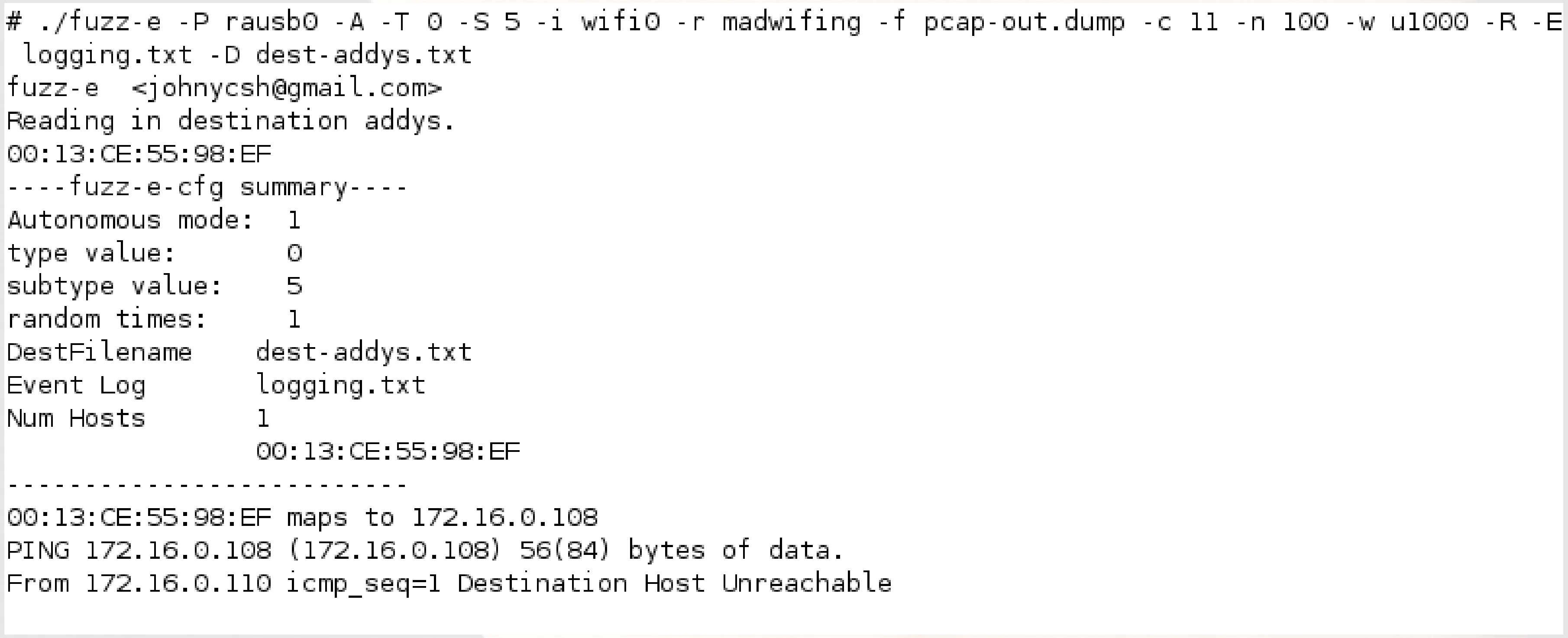
```
08 02 D5 00 00 02 6F 33 46 84 00 0C 41 3F 31 3E ......o3F...A?1>
00 0C 41 3F 31 3E 20 1E AA AA 03 00 00 00 88 8E ..A?1> .........
01 03 00 77 FE 01 C9 00 20 00 00 00 00 00 00 00 ...w.... .......
01 22 38 F9 B2 12 82 DB FD 2A 7A 3C A4 CE 9B F8 ."8......*z<....
82 6D FA 39 8A 6D 47 9D 96 9D 68 01 E8 76 82 83 .m.9.mG...h..v..
C9 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................
00 64 3B 13 38 8F 12 20 1E 06 0A 58 7A 6E FE 29 .d;.8.. ...Xzn.)
E8 FF FF FF 16 00 50 F2 01 01 00 00 50 F2 02 01 ......P.....P...
00 00 50 F2 [illegible] 01 00 00 50 F2 02          ..P.....P..
```

What happens when the WPA key length is 0xffff?

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

Airbase

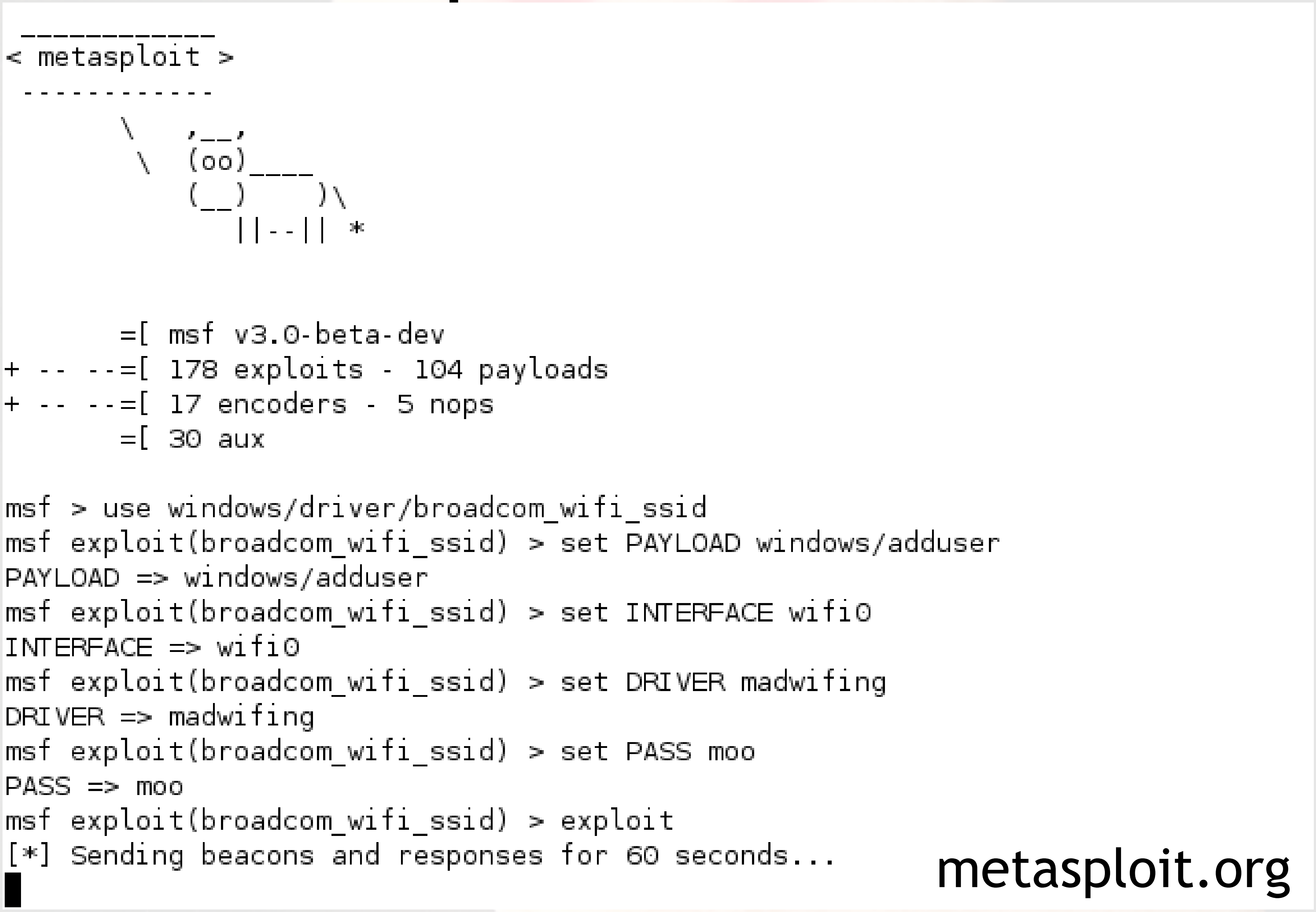
- Collection of tools for manipulating wireless networks (rock on Johnycsh!)
- Fuzzers, accelerated WEP cracking, frame manipulation tools, oh my!



```
# ./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  <johnycsh@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



```
 ____________
< metasploit >
 ------------
       \   ,__,
        \  (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...
█
```

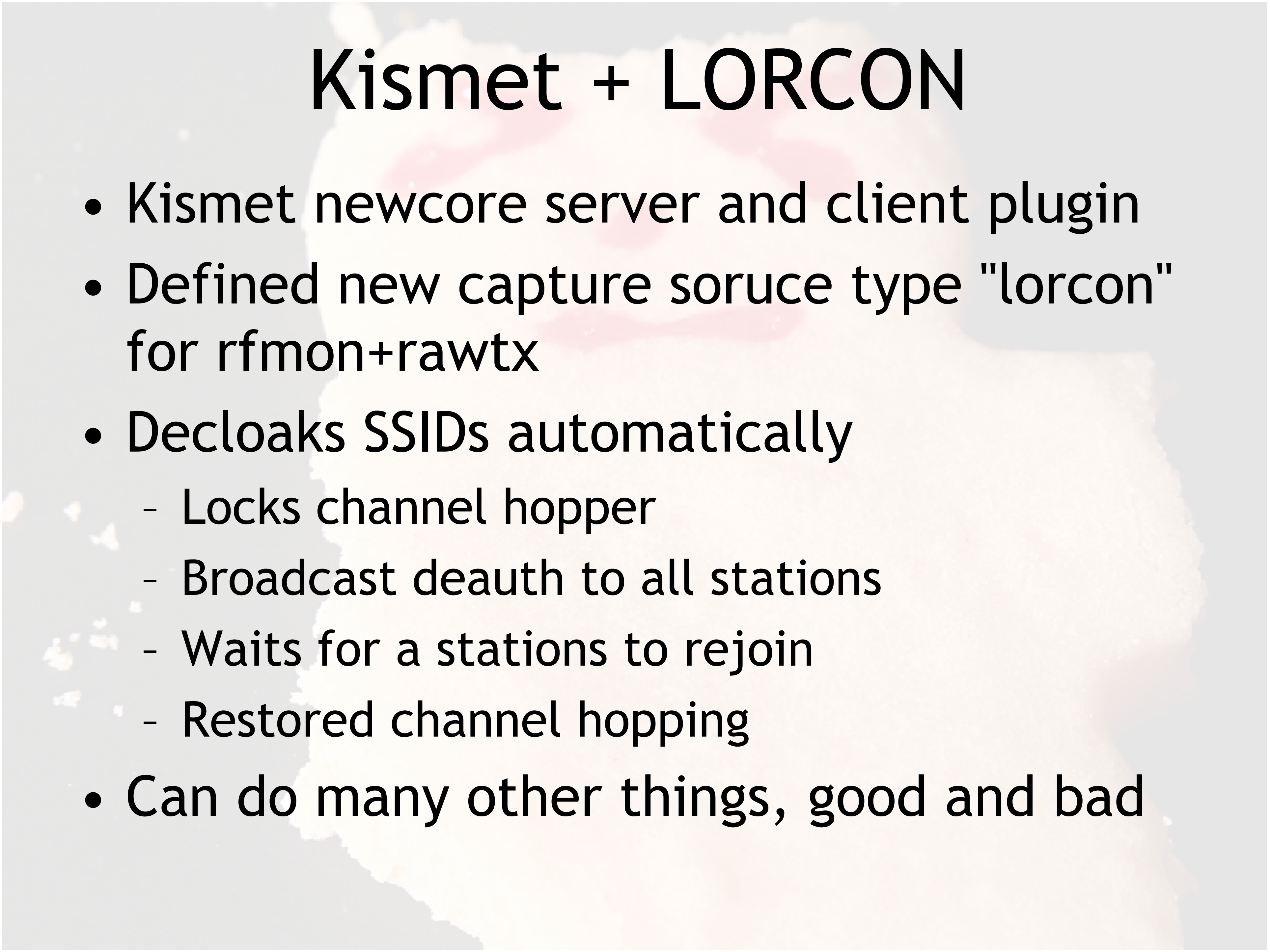
metasploit.org

Ruby + LORCON

- Ruby module from the Metasploit Framework

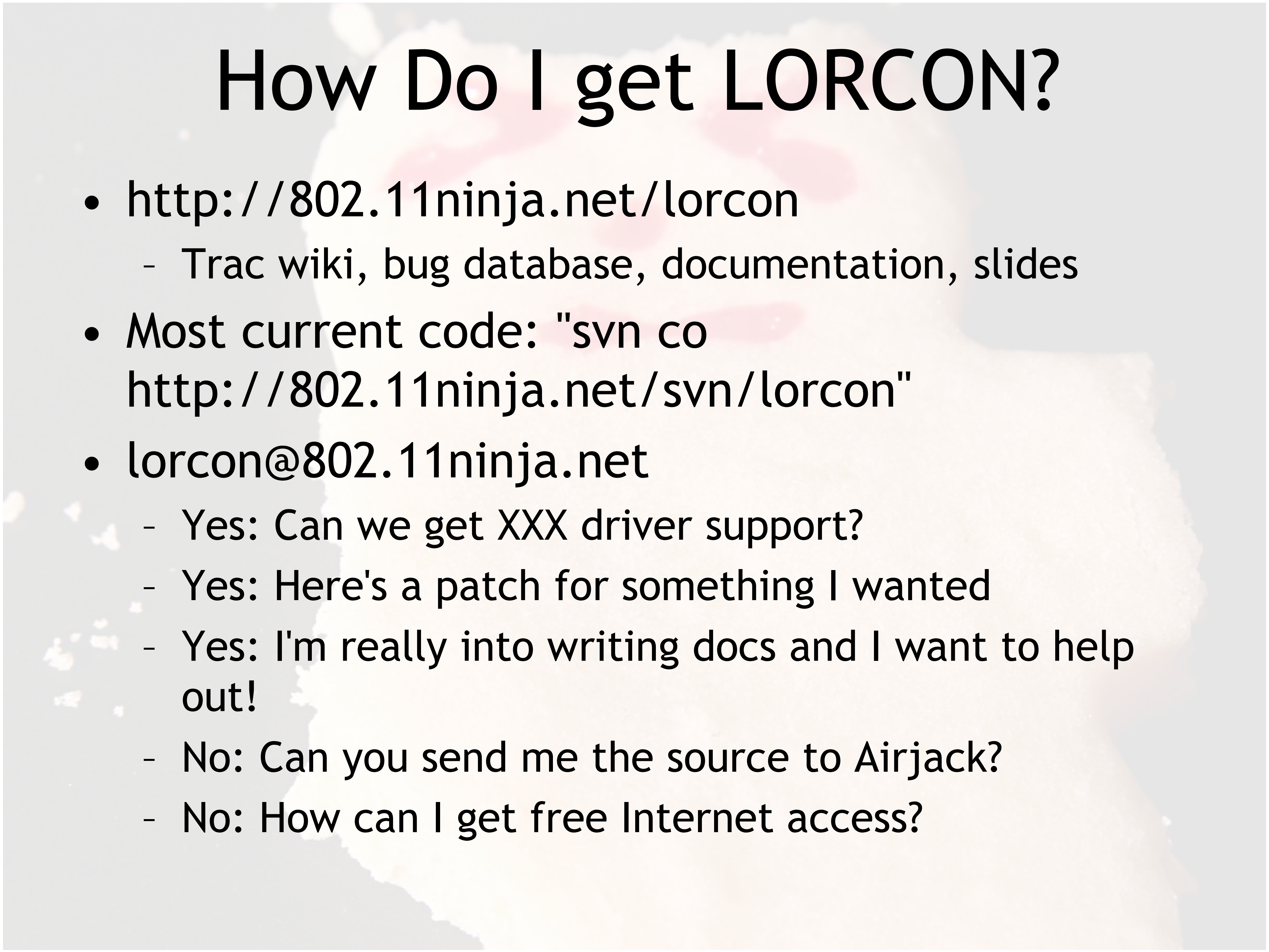
```
require "Lorcon"
packet=[0xc4,0x00,0xff,0x7f,0x00,0x13,0xce,0x55,0x98,0xef].pack('C*')

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"
```

```
$ 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 soruce type "lorcon" for rfmon+rawtx
- Decloaks SSIDs automatically
  - Locks channel hopper
  - Broadcast deauth to all stations
  - Waits for a 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