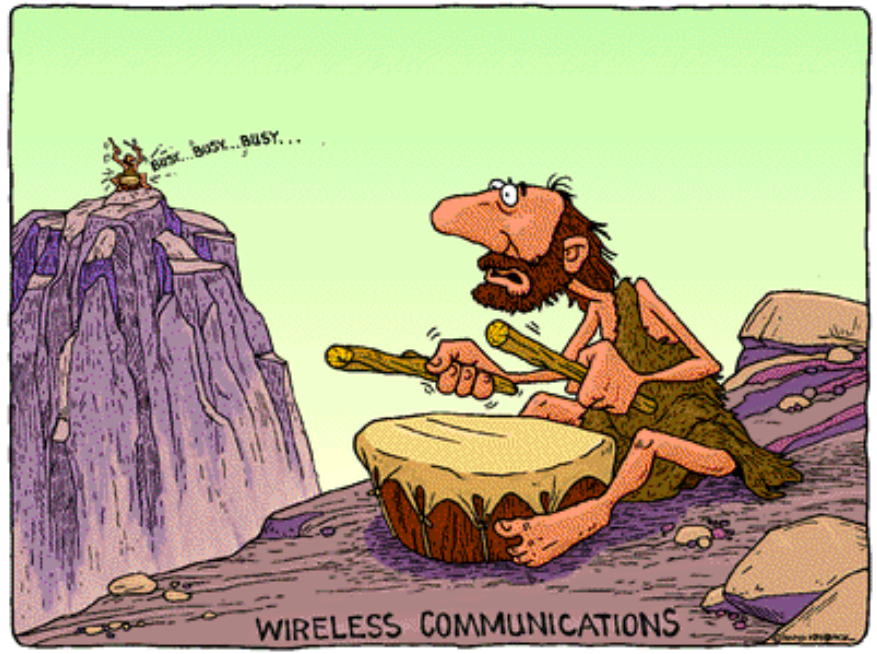
# The Need for an 802.11 Wireless Toolkit

***Invictus Ferramenta***

Mike Schiffman

BlackHat Briefings July 2002

Version 2.0




@stake

Where Security & Business Intersect℠

# Agenda

1. Introduction and Overview
2. Protocol Primer
3. Protocol Flaws
4. Existing Tools and Gap Analysis
5. Radiate
6. Theory into Practice: Radiate and Libnet
7. Closing Comments and Questions

# Mike Schiffman

- Senior Consultant with @stake
  - *The Premier provider of Digital Security Services*
- Technical Advisory Board for Qualys, Inc.
- Consulting Editor for Wiley & Sons
- R&D background
  - Firewalk, Libnet, Libsf, Libradiate, Phrack Magazine
- Books:
  - Building Open Source Network Security Tools, Wiley & Sons
  - Hacker's Challenge I, Osborne McGraw-Hill
  - Hacker's Challenge II, Osborne McGraw-Hill

# Overview

- What you will learn
  - Brief intro to the 802.11 protocol
  - Weaknesses in the 802.11 protocol
  - How to use libradiate to build custom 802.11 security tools
- What you should know
  - General understanding of the  TCP/IP protocol suite
    - Primarily layers 2 - 3
  - General understanding of wireless
  - General network security concepts
  - The C programming language

# Nomenclature

- Network Security Tool or Tool
  - *A network security tool is an algorithmic implement that is designed to probe, assess, or increase the overall safety of or mitigate risk associated with an entity across a communications medium.*
  - This is dry but it works
- Toolkit
  - An API, or set of APIs used to build Network Security Tools
  - A C programming library
  - “Component”

# 802.11 is Everywhere

- 802.11-based networks are wonderful inventions
  - Corporate America
  - Coffee shops, Hotels, Airports, “Residential ISPs”
- Many new products and services on top of 802.11
  - Newer, faster physical interfaces being turned out on top of the same layer 2 protocols
- However, there are a “few” security concerns…
  - We need a way to be able to test for security issues
  - Sure, some tools do exist
- But what we really need is a way to be able to test for arbitrary security issues with custom tools
  - We need a generic 802.11 toolkit

# 802.11 Primer

- Borne out of the IEEE 802 LMSC
- 802.11 WLAN standard
  - PHY layer: 802.11b 2.4Ghz up to 11Mbps
  - PHY layer: 802.11a 5Ghz up to 54Mbps
- Drop in replacement for Ethernet
  - Upper layer protocols should be none the wiser
  - This seamless integration comes at a stiff price – under the hood complexity

# 802.11 Primer: Physical Interface

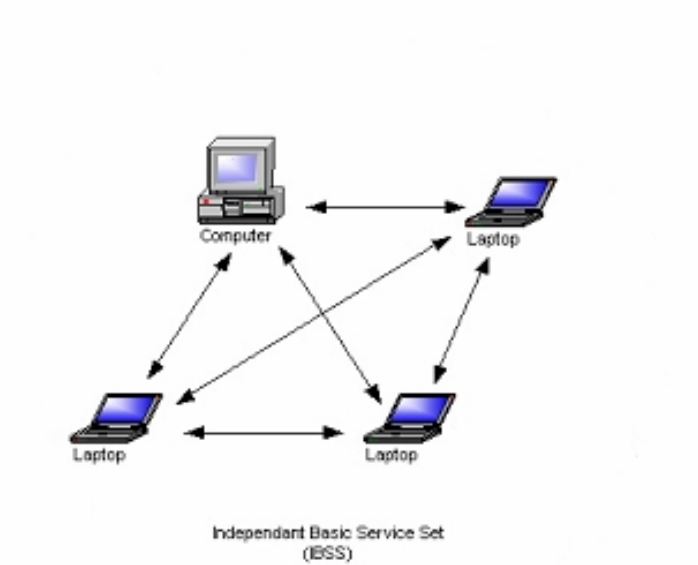
- 802.11b is the most widely deployed
- Direct Sequence Spread Spectrum (DSSS)
  - 2.4GHz ISM Band
    - Industrial / Instrumentation, Scientific, Medical
    - 2.400GHz – 2.4835GHz
    - 14 channels or frequency divisions
      - 1 – 11 used in the United States
  - 1000mW power maximum
    - Most devices are 30mW – 100mW

# 802.11 Primer: MAC Sublayer Tidbits

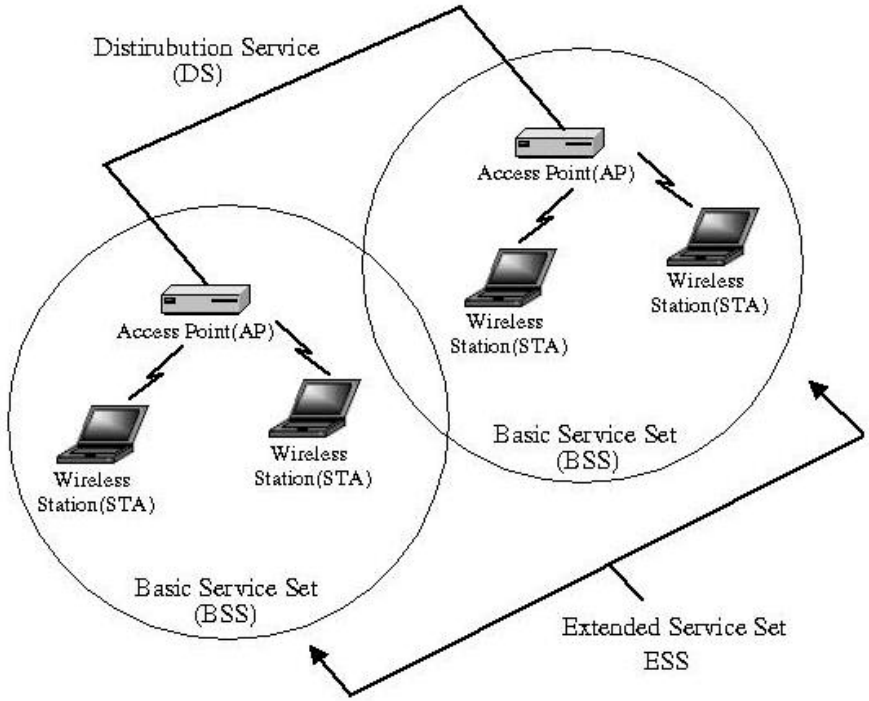
- CSMA/CA
  - LBT (Listen Before Talk)
  - Exponential back off and retry
  - Collision avoidance via physical carrier sense and Network Allocation Vector (NAV)
    - Sent in most frames (duration ID)
    - Informs other stations how long the medium will be in-use
    - Virtual carrier sense (station waits until NAV is 0 before attempting to send)

# Configuration Options

## AD Hoc




## Infrastructure



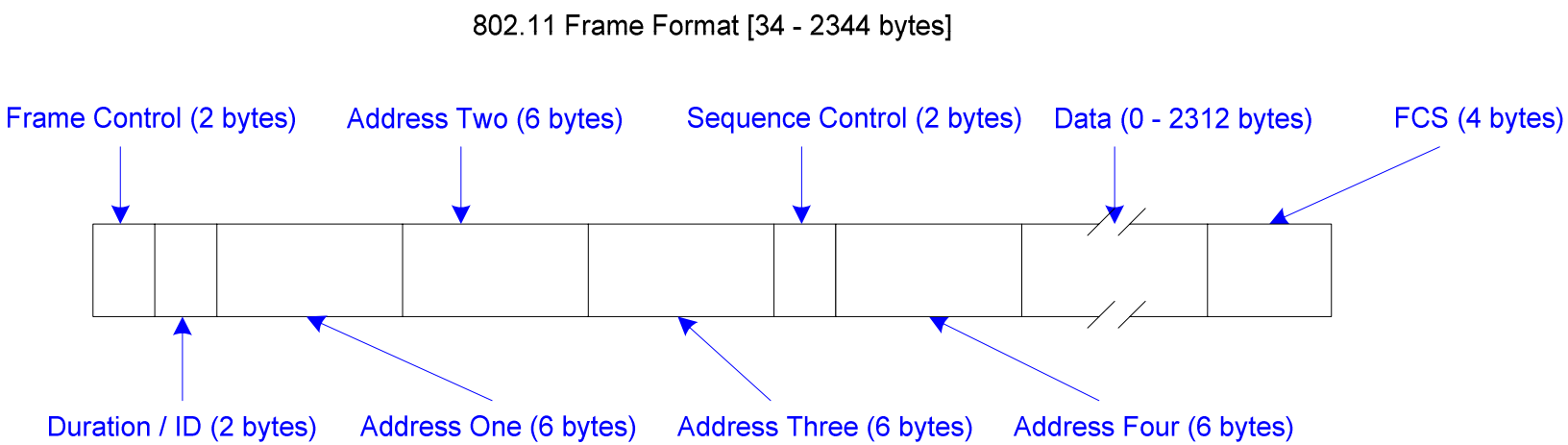
# Some Management Frame Subtypes

- Beacon
  - Transmitted frequently announcing availability and capabilities of BSS (Infrastructure Mode)
- Probe Request and Response
  - Client initiated request for a WLAN
  - Response is essentially the same as a beacon
- Associate Request and Response
  - “I’d like to be a part of your BSS”
- Disassociate
  - Tell your story walking!

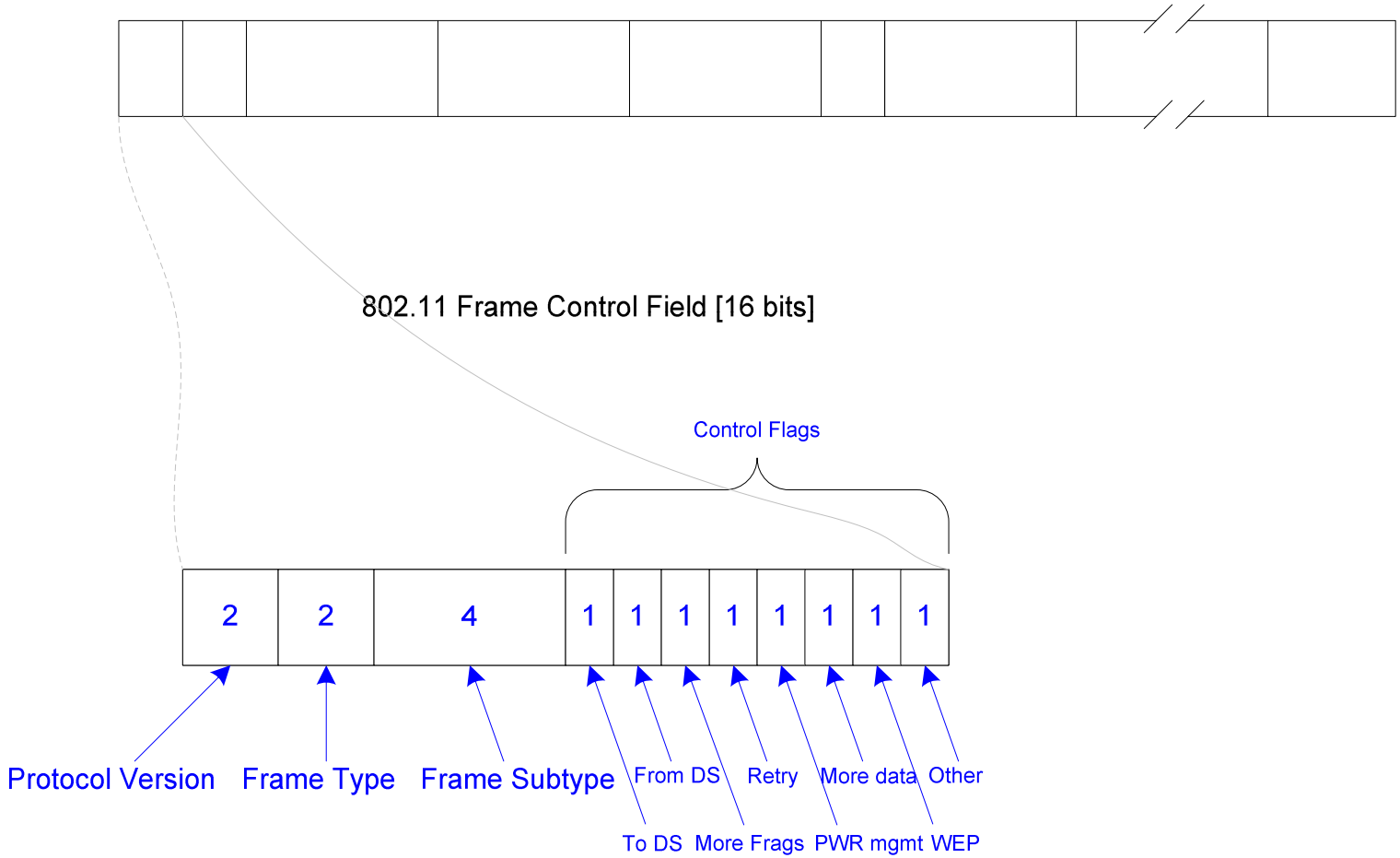
# Some Control Frame Subtypes

- RTS
  - “I’d like to send a frame or two”
  - Updates NAV values for neighboring stations (transmitter)
- CTS
  - “Sounds good”
  - Updates NAV values for neighboring stations (receiver)
- ACK
  - “Got your data”
  - Also updates NAV as per CTS

# 802.11 Frame Layout

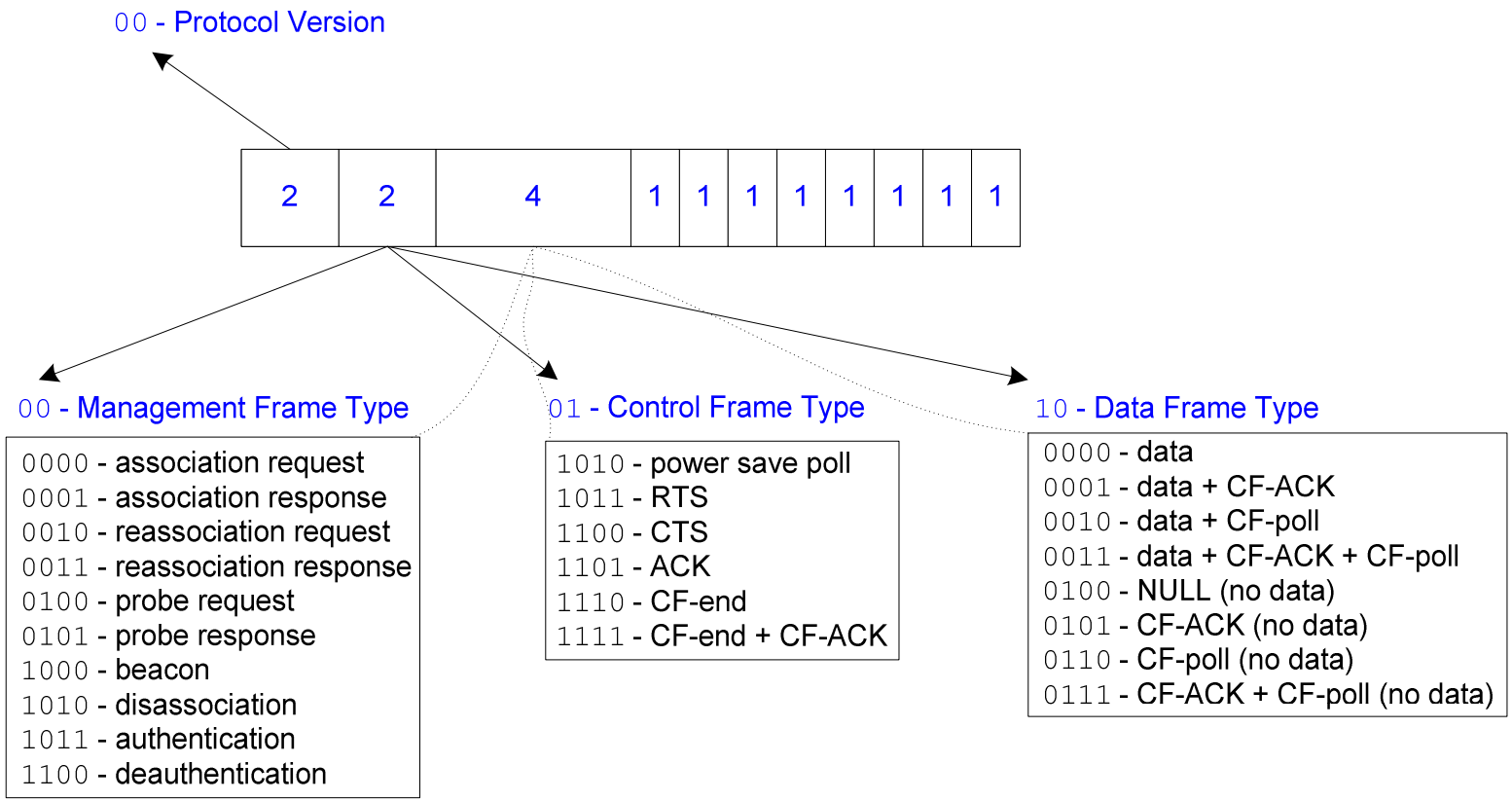


# 802.11 Frame Control Field



# 802.11 Types and Subtypes

802.11 Types and Subtypes



# A Grip of 802.11 Protocol Weaknesses

- Wired Equivalent Privacy (WEP)
- Management and Control Frames
  - Probe request and replies
  - No authentication or encryption
- Vendor Specific Issues
  - State maintenance
  - Fringe frames / packets

# WEP Weaknesses

- Keysizes
- Weak IVs
- Growing keyspace via Induction
- Key Management Issues
  - Revocation and Expiry
  - Distribution
  - Rotation

# 802.11 Management Weaknesses

- Neither encrypted nor authenticated
  - Information leakage
    - Access point and client enumeration
  - Spoofing
    - Denial of Service
    - Association thievery
    - Data injection
  - Flooding
    - Denial of Service

# 802.11 Vendor Specific Weaknesses

- Every device is different often from a separate codebase
  - Different IP stacks
  - Different sizes of state tables
    - Flooding
  - Different sanity checks on frames / packets
    - Fringe frames / packets
  - Different devices have different infrastructure
    - SNMP
    - telnet

# 802.11 Weaknesses Summary

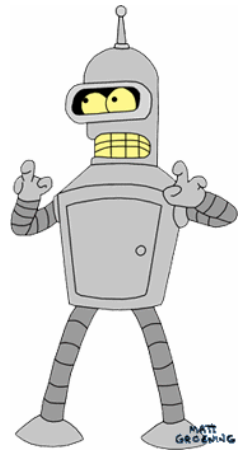
- It's a new protocol
- All of these are problems we've seen before
  - Exacerbated by the fact that a shared wireless medium is inherently insecure
- A myriad of tools exist to test for similar vulnerabilities on wired networks…
  - There are several toolkits available
    - Libnet
    - Libdnet
    - Libpcap

# Existing 802.11 Network Security Tools

- AiroPeek / AiroPeek NX
  - Wireless frame sniffer / analyzer
- AirTraf
  - Wireless sniffer / analyzer / “IDS”
- AirSnort
  - WEP key “cracker”
- NetStumbler
  - Access point enumeration tool

# Where Existing Tools Come Up Lacking

- Very task-oriented and specific
- Closed-source tools are not tunable
  - “I’d like to use NetStumbler to listen for Beacons”
  - TOUGH LUCK DORKUS!

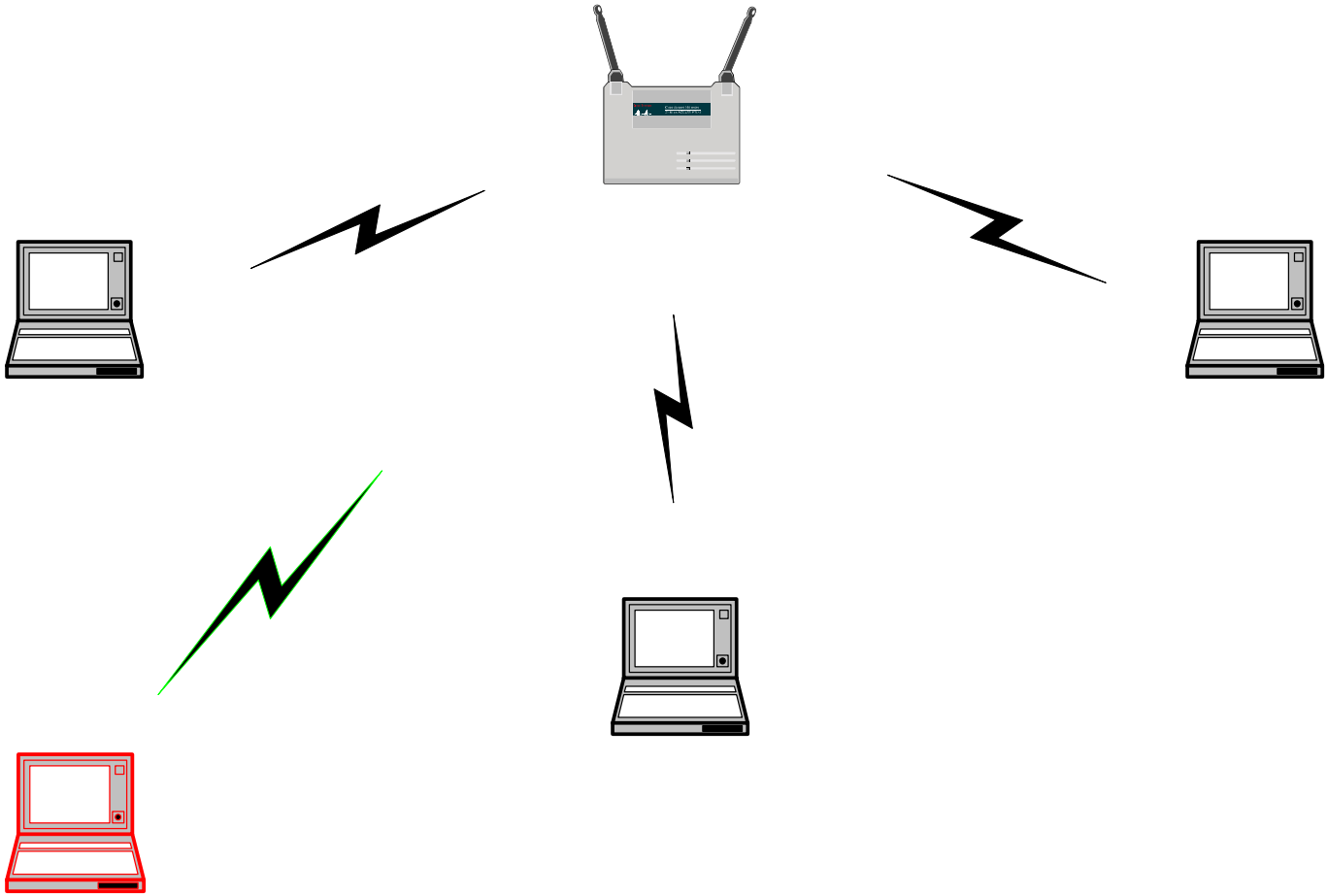


- In order to test for unspecified security issues there needs to be a generic testing framework…

# Case Study: The Byzantine Generals Problem

- It is important for an 802.11 network to be robust
  - Strongly formed or constructed
- Byzantine Robustness means proof against Byzantine failure
  - Byzantine generals problem

# Omerta (Byzantine Fault Injection)



# Byzantine Failure and the Need for a Toolkit

- Byzantine fault injection
  - Omerta in practice
  - Theories are nice, code is better
- In order to ferret out Byzantine failure vulnerabilities requires arbitrary security tools
- A generic toolkit would allow an application developer to build tools to test for Byzantine failures via controlled Byzantine fault injection
- One example of such a toolkit for wired networks would be our handsome friend Libnet…

# Omerta Frame Dump

| Packet | Source | Destination | BSSID | Channel | Flags | Size | Absolute Time | Protocol |
|---|---|---|---|---|---|---|---|---|
| 1 | 00:40:96:54:56:33 | Broadcast | 00:40:96:54:56:33 | 11 | * | 68 | 10:33:32.081303 | 802.11 Beacon |
| 2 | 00:40:96:54:56:33 | Broadcast | 00:40:96:54:56:33 | 11 | * | 68 | 10:33:32.184083 | 802.11 Beacon |
| 3 | 00:40:96:54:56:33 | Broadcast | 00:40:96:54:56:33 | 11 | * | 68 | 10:33:32.286014 | 802.11 Beacon |
| 4 | 00:40:96:54:56:33 | Broadcast | 00:40:96:54:56:33 | 11 | * | 68 | 10:33:32.388512 | 802.11 Beacon |
| 5 | 00:40:96:54:56:33 | Broadcast | 00:40:96:54:56:33 | 11 | * | 68 | 10:33:32.490692 | 802.11 Beacon |
| 6 | 00:40:96:54:56:33 | Broadcast | 00:40:96:54:56:33 | 11 | * | 68 | 10:33:32.593509 | 802.11 Beacon |
| 7 | 00:40:96:44:17:DF | 00:07:50:57:E4:7B | 00:40:96:54:56:33 | 11 | | 104 | 10:33:39.587208 | 802.11 WEP Data |
| 8 | | 00:40:96:44:17:DF | | 11 | # | 14 | 10:33:39.587322 | 802.11 Ack |
| 9 | 00:40:96:44:17:DF | 00:07:50:57:E4:7B | 00:40:96:54:56:33 | 11 | * | 30 | 10:33:39.588392 | 802.11 Disassoc |
| 10 | | 00:40:96:44:17:DF | | 11 | # | 14 | 10:33:39.588604 | 802.11 Ack |
| 11 | 00:40:96:44:17:DF | 00:07:50:57:E4:7B | 00:40:96:54:56:33 | 11 | * | 30 | 10:33:39.590059 | 802.11 Disassoc |
| 12 | 00:40:96:54:56:33 | Broadcast | 00:40:96:54:56:33 | 11 | | 72 | 10:33:39.590706 | 802.11 WEP Data |
| 13 | 00:40:96:44:17:DF | 00:07:50:57:E4:7B | 00:40:96:54:56:33 | 11 | * | 30 | 10:33:39.592594 | 802.11 Disassoc |
| 14 | | 00:40:96:44:17:DF | | 11 | # | 14 | 10:33:39.592803 | 802.11 Ack |
| 15 | 00:40:96:44:17:DF | 00:07:50:57:E4:7B | 00:40:96:54:56:33 | 11 | * | 30 | 10:33:39.594403 | 802.11 Disassoc |
| 16 | | 00:40:96:44:17:DF | | 11 | # | 14 | 10:33:39.594613 | 802.11 Ack |
| 17 | 00:40:96:44:17:DF | 00:07:50:57:E4:7B | 00:40:96:54:56:33 | 11 | * | 30 | 10:33:39.604654 | 802.11 Disassoc |
| 18 | | 00:40:96:44:17:DF | | 11 | # | 14 | 10:33:39.604864 | 802.11 Ack |
| 19 | 00:40:96:44:17:DF | 00:07:50:57:E4:7B | 00:40:96:54:56:33 | 11 | * | 30 | 10:33:39.606692 | 802.11 Disassoc |
| 20 | | 00:40:96:44:17:DF | | 11 | # | 14 | 10:33:39.606906 | 802.11 Ack |
| 21 | 00:40:96:44:17:DF | 00:07:50:57:E4:7B | 00:40:96:54:56:33 | 11 | * | 30 | 10:33:39.625889 | 802.11 Disassoc |
| 22 | | 00:40:96:44:17:DF | | 11 | # | 14 | 10:33:39.626100 | 802.11 Ack |
| 23 | 00:40:96:44:17:DF | 00:07:50:57:E4:7B | 00:40:96:54:56:33 | 11 | * | 30 | 10:33:39.628664 | 802.11 Disassoc |
| 24 | | 00:40:96:44:17:DF | | 11 | # | 14 | 10:33:39.628875 | 802.11 Ack |
| 25 | 00:40:96:44:17:DF | 00:07:50:57:E4:7B | 00:40:96:54:56:33 | 11 | | 104 | 10:33:40.104549 | 802.11 WEP Data |
| 26 | | 00:40:96:44:17:DF | | 11 | # | 14 | 10:33:40.104670 | 802.11 Ack |
| 28 | 00:40:96:54:56:33 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 11 | * | 30 | 10:33:40.106577 | 802.11 Disassoc |
| 29 | | 00:40:96:54:56:33 | | 11 | # | 14 | 10:33:40.106788 | 802.11 Ack |
| 30 | 00:40:96:44:17:DF | 00:07:50:57:E4:7B | 00:40:96:54:56:33 | 11 | * | 30 | 10:33:40.107924 | 802.11 Disassoc |

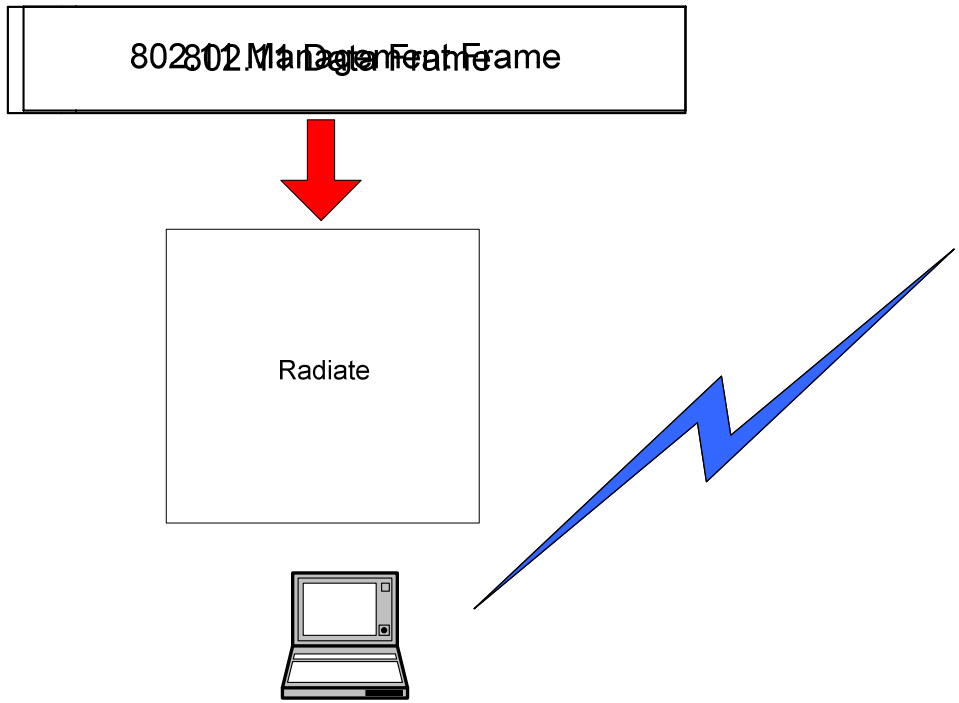
# Omerta Frame Dump

| Packet | Source | Destination | BSSID | Channel | Flags | Size | Absolute Time | Protocol |
|---|---|---|---|---|---|---|---|---|
| 1 | | 00:40:96:44:17:DF | | 11 | # | 14 | 10:33:40.130619 | 802.11 Ack |
| 2 | 00:40:96:44:17:DF | Broadcast | Broadcast | 11 | * | 47 | 10:33:40.218892 | 802.11 Probe Req |
| 3 | 00:40:96:54:56:33 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 11 | * | 62 | 10:33:40.219727 | 802.11 Probe Rsp |
| 4 | 00:40:96:54:56:33 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 11 | * | 62 | 10:33:40.221203 | 802.11 Probe Rsp |
| 5 | 00:40:96:44:17:DF | Broadcast | Broadcast | 11 | * | 47 | 10:33:40.235273 | 802.11 Probe Req |
| 6 | 00:40:96:54:56:33 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 11 | * | 62 | 10:33:40.236116 | 802.11 Probe Rsp |
| 7 | | 00:40:96:54:56:33 | | 11 | # | 14 | 10:33:40.236402 | 802.11 Ack |
| 8 | 00:40:96:44:17:DF | Broadcast | Broadcast | 11 | * | 47 | 10:33:40.290570 | 802.11 Probe Req |
| 9 | 00:40:96:44:17:DF | Broadcast | Broadcast | 11 | * | 47 | 10:33:40.306960 | 802.11 Probe Req |
| 10 | 00:40:96:54:56:33 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 11 | * | 62 | 10:33:40.307792 | 802.11 Probe Rsp |
| 11 | | 00:40:96:54:56:33 | | 11 | # | 14 | 10:33:40.308077 | 802.11 Ack |
| 12 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 00:40:96:54:56:33 | 11 | * | 57 | 10:33:40.345170 | 802.11 Reassoc Req |
| 13 | | 00:40:96:44:17:DF | | 11 | # | 14 | 10:33:40.345453 | 802.11 Ack |
| 14 | 00:40:96:54:56:33 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 11 | * | 84 | 10:33:40.349009 | 802.11 Reassoc Rsp |
| 15 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 00:40:96:54:56:33 | 11 | | 61 | 10:33:40.350153 | 802.11 WEP Data |
| 16 | | 00:40:96:44:17:DF | | 11 | # | 14 | 10:33:40.350448 | 802.11 Ack |
| 17 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 00:40:96:54:56:33 | 11 | * | 30 | 10:33:40.351239 | 802.11 Disassoc |
| 18 | | 00:40:96:44:17:DF | | 11 | # | 14 | 10:33:40.351449 | 802.11 Ack |
| 19 | 00:40:96:54:56:33 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 11 | | 94 | 10:33:40.352185 | 802.11 WEP Data |
| 20 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 00:40:96:54:56:33 | 11 | * | 30 | 10:33:40.352559 | 802.11 Disassoc |
| 21 | | 00:40:96:44:17:DF | | 11 | # | 14 | 10:33:40.352769 | 802.11 Ack |
| 22 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 00:40:96:54:56:33 | 11 | C | 118 | 10:33:40.354109 | 802.11 WEP Data |
| 23 | 00:40:96:54:56:33 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 11 | | 118 | 10:33:40.354873 | 802.11 WEP Data |
| 24 | | 00:40:96:54:56:33 | | 11 | # | 14 | 10:33:40.354986 | 802.11 Ack |
| 25 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 00:40:96:54:56:33 | 11 | | 118 | 10:33:40.356639 | 802.11 WEP Data |
| 26 | | 00:40:96:44:17:DF | | 11 | # | 14 | 10:33:40.356896 | 802.11 Ack |
| 27 | 00:40:96:54:56:33 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 11 | * | 30 | 10:33:40.358420 | 802.11 Disassoc |
| 28 | | 00:40:96:54:56:33 | | 11 | # | 14 | 10:33:40.358631 | 802.11 Ack |
| 29 | 00:40:96:44:17:DF | 00:40:96:54:56:33 | 00:40:96:54:56:33 | 11 | * | 30 | 10:33:40.359179 | 802.11 Disassoc |

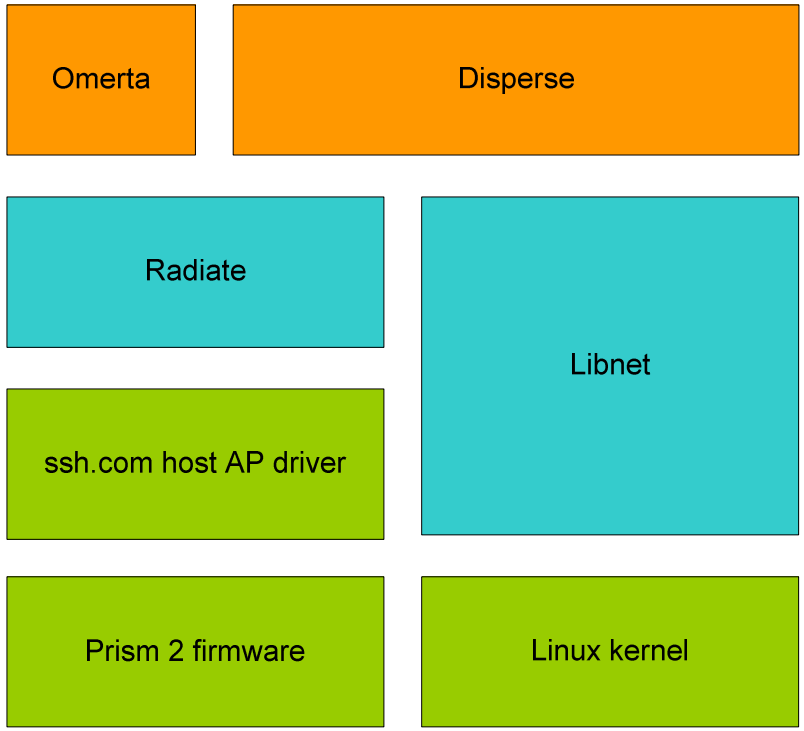
# (lib)Radiate

- The 802.11 wireless analog (addition?) to Libnet
- Very early beta!
- 802.11 frame capturing, creation and injection library
- Simple C library consisting of a few function calls and a header file
- Version 0.3 currently runs only under Intersil Prism2-based cards
  - SMC, D-link, etc
  - Prism-2.5 support?

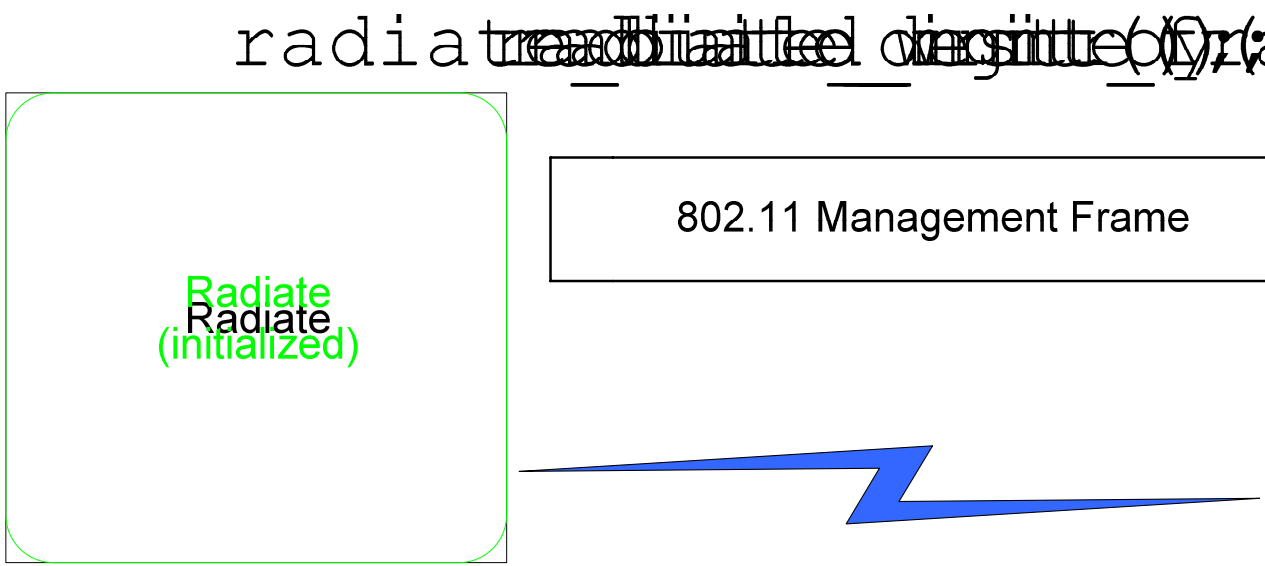
# Radiate 50,000 Foot View



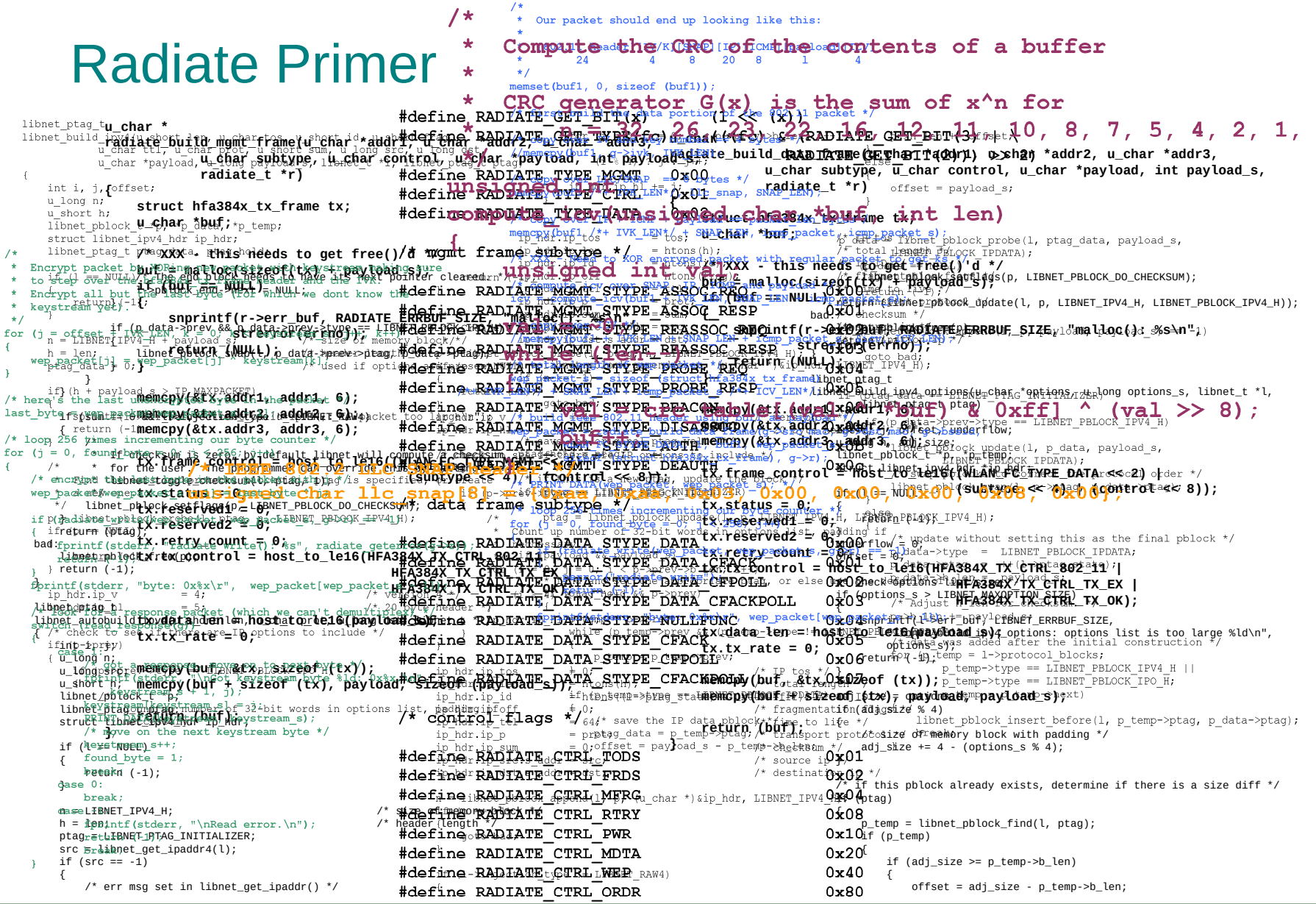
# Radiate and Related Component Relationships



# Radiate 25,000 Foot View



# Radiate Primer



# Radiate Context and Framework Functions

```
struct radiate_context
{
    int fd;
    u_char flags;
#define RADIATE_VERBOSE 0x1         /* verbose debug information */
    char err_buf[RADIATE_ERRBUF_SIZE];
};
typedef struct radiate_context radiate_t;
```

```
radiate_t *radiate_init(u_char control_flags, char *err_buf);

void radiate_destroy(radiate_t *r);
```

# Construction Functions

```
u_char *
       radiate_build_mgmt_frame(u_char *a1, u_char *a2, u_char *a3,
       u_char subtype, u_char control, u_char *payload, int payload_s,
       radiate_t *r);

u_char *
       radiate_build_data_frame(u_char *a1, u_char *a2, u_char *a3,
       u_char subtype, u_char control, u_char *payload, int payload_s,
       radiate_t *r);
```

# Construction Function Symbolic Constants

```
/* management subtypes */
RADIATE_MGMT_STYPE_ASSOC_REQ    /* association request */
RADIATE_MGMT_STYPE_ASSOC_RESP   /* association response */
RADIATE_MGMT_STYPE_REASSOC_REQ  /* reassociation request */
RADIATE_MGMT_STYPE_REASSOC_RESP /* reassociation response */
RADIATE_MGMT_STYPE_PROBE_REQ    /* probe request */
RADIATE_MGMT_STYPE_PROBE_RESP   /* probe response */
RADIATE_MGMT_STYPE_BEACON       /* beacon */
RADIATE_MGMT_STYPE_DISASSOC     /* disassociaion */
RADIATE_MGMT_STYPE_AUTH         /* authentication */
RADIATE_MGMT_STYPE_DEAUTH       /* deauthentication */

/* data subtypes */
RADIATE_DATA_STYPE_DATA         /* just data */
RADIATE_DATA_STYPE_DATA_CFACK   /* data + contention free + ACK */
RADIATE_DATA_STYPE_DATA_CFPOLL  /* data + contention free + poll */
RADIATE_DATA_STYPE_DATA_CFACKPOLL  /* data + contention free + ACK/poll */
RADIATE_DATA_STYPE_NULLFUNC     /* NULL function (no data) */
RADIATE_DATA_STYPE_CFACK        /* contention free + ACK */
RADIATE_DATA_STYPE_CFPOLL       /* contention free + poll */
RADIATE_DATA_STYPE_CFACKPOLL    /* contention free + ACK/poll */

/* control bits, logical OR together to combine */
RADIATE_CTRL_TODS               /* to DS */
RADIATE_CTRL_FRDS               /* from DS */
RADIATE_CTRL_MFRG               /* more fragments */
RADIATE_CTRL_RTRY               /* retry */
RADIATE_CTRL_PWR                /* power management */
RADIATE_CTRL_MDTA               /* more data */
RADIATE_CTRL_WEP                /* WEP */
RADIATE_CTRL_ORDR               /* order */
```

# Capture and Injection Functions

```
int radiate_read(u_char **buf, radiate_t *r);

int radiate_write(u_char *frame, int frame_s, radiate_t *r);

struct hfa384x_rx_frame {
        /* HFA384X RX frame descriptor */
        u16 status; /* HFA384X_RX_STATUS_ flags */
        u32 time; /* timestamp, 1 microsecond resolution */
        u8 silence; /* 27 .. 154; seems to be 0 */
        u8 signal; /* 27 .. 154 */
        u8 rate; /* 10, 20, 55, or 110 */
        u8 rxflow;
        u32 reserved;

        /* 802.11 */
        u16 frame_control;
        u16 duration_id;
        u8 addr1[6];
        u8 addr2[6];
        u8 addr3[6];
        u16 seq_ctrl;
        u8 addr4[6];
        u16 data_len;

        /* 802.3 */
        u8 dst_addr[6];
        u8 src_addr[6];
        u16 len;

        /* followed by frame data; max 2304 bytes */
} __attribute__ ((packed));
```

# Miscellaneous Functions

```
int radiate_set_mm(char *mode, radiate_t *r);

char *radiate_geterror(radiate_t *r);
```

# Radiate Initialization

- Radiate Context
  - Communication with the kernel via a netlink socket
  - Maintains state

```
radiate_t *r;
u_char control_flags = 0;

r = radiate_init(control_flags, errbuf);
if (r == NULL)
{
    fprintf(stderr, “radiate_init(): %s”, errbuf);
}
```

# Radiate Data Frame Creation

- Implicit `malloc(3)`

```
u_char *buf;
u_char data = “packet payload”;

buf = radiate_build_data_frame(mac1, mac2, bssid, RADIATE_DATA_STYPE_DATA, 0, data, strlen(data), r);
if (buf == NULL)
{
    fprintf(stderr, “radiate_build_data_frame(): %s\n”, radiate_geterror(r));
    goto bad;
}
```

# Radiate Management Frame Creation

- Implicit `malloc(3)`

```
u_char *buf;
u_char data = RADIATE_REASON_PREV_AUTH_NOT_VALID;

buf = radiate_build_mgmt_frame(mac1, mac2, bssid, RADIATE_MGMT_STYPE_DISASSOC, 0,
            (u_char *)&data, sizeof (data), r);
if (buf == NULL)
{
    fprintf(stderr, “radiate_build_mgmt_frame(): %s\n”, radiate_geterror(r));
    goto bad;
}
```

# Radiate Frame Capture

- Blocking read via `recv(2)`
  - Buffer will contain an 802.11 frame with an `hfa384x_rx_frame` header prepended

```
u_char *buf;
int c;

c = radiate_read(&buf, r);
if (c == -1)
{
    fprintf(stderr, "radiate_read(): %s\n", radiate_geterror(r));
    goto bad;
}
```

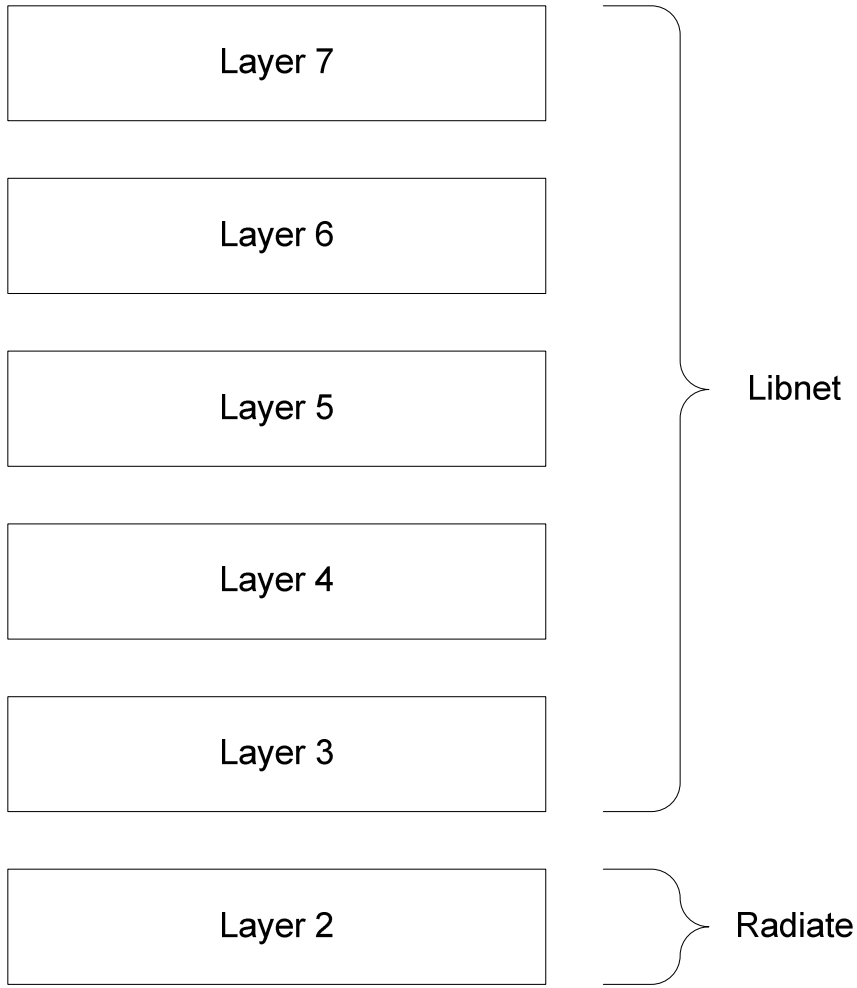
# Radiate Frame Injection

- Write via `send(2)`
  - Buffer will contain an 802.11 frame with an `hfa384x_rx_frame` header prepended – radiate handles the semantics

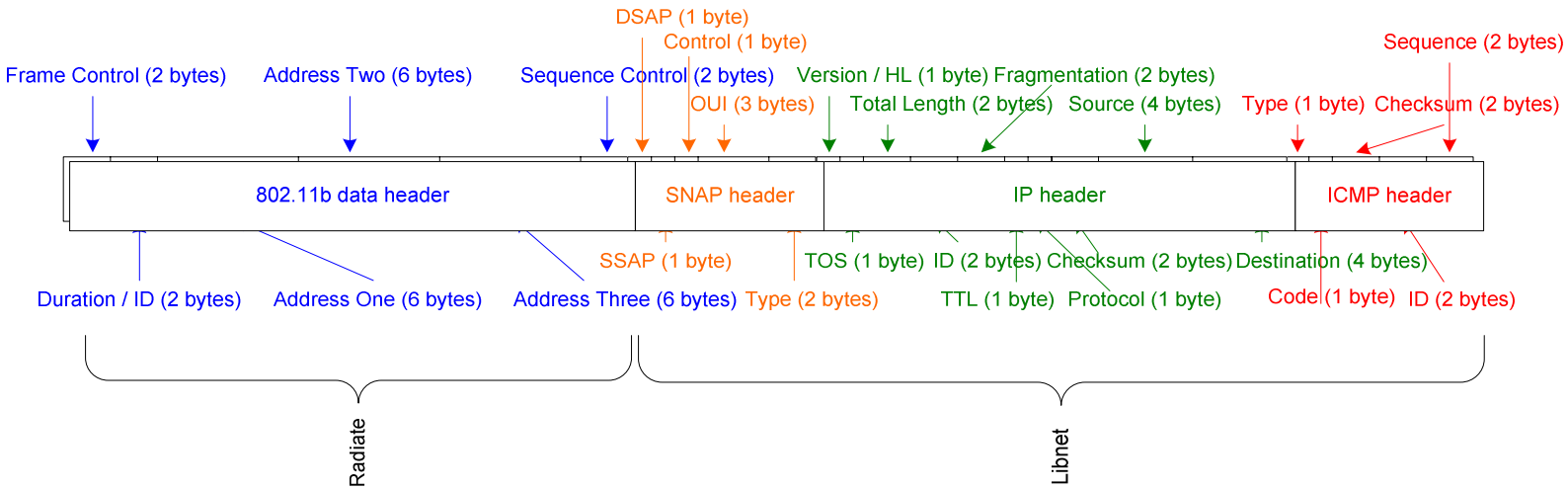
```
int c;

c = radiate_write(frame, frame_s, r);
if (c == -1)
{
    fprintf(stderr, “radiate_write(): %s\n”, radiate_geterror(r));
    goto bad;
}
```

# Radiate and Libnet – Two Best Pals!

- Use Libnet to build the Layer 3 and above headers
- The Arbaugh Inductive attack was implemented using Radiate
  - Remember: Theories are nice, Code is better!



# Theory into Practice…



# Using Libradiate and Libnet

- *Disperse*
- Small 300 line program (with copious comments)
- Uses libradiate and libnet to build and send an 802.11 encapsulated ICMP ECHO packet

# disperse.h

```
#include <errno.h>
#include <time.h>
#include <libnet.h>
#include <radiate.h>

void do_ioctl(char *, radiate_t *r);
void usage(char *);

/* IEEE 802.2 LLC SNAP header */
unsigned char llc_snap[8] = {0xaa, 0xaa, 0x03, 0x00, 0x00, 0x00, 0x08, 0x00};

struct disperse_pack
{
    radiate_t *r;                       /* radiate context */
    libnet_t *l;                        /* libnet context */
    u_char protocol;                    /* protocol */
    u_char *frame;                      /* frame pointer */
    u_char *dst_mac;                    /* destination MAC address */
    u_char *src_mac;                    /* source MAC address */
    u_char *bssid;                      /* BSSID address */
    u_long dst_ip;                      /* destination IP address */
    u_long src_ip;                      /* source IP address */
    u_char *payload;                    /* packet payload */
};
```

# disperse.c, Libnet Initialization

```
#include "disperse.h"

int
main(int argc, char **argv)
{
    int c, unused;
    libnet_ptag_t ptag;
    struct disperse_pack *dp;
    u_char *ip_packet, *payload;
    u_long ip_packet_s, payload_s;
    char err_buf[LIBNET_ERRBUF_SIZE];

    printf("Disperse [Libradiate sample code]\n");

    dp = malloc(sizeof (struct disperse_pack));
    if (dp == NULL)
    {
        fprintf(stderr, "malloc() %s\n", strerror(errno));
        return (EXIT_FAILURE);
    }
    memset(dp, 0, sizeof (struct disperse_pack));

    /* initialize our dummy libnet context, needed for packet construction */
    dp->l = libnet_init(
            LIBNET_RAW4_ADV,                        /* injection type */
            NULL,                                   /* network interface */
            err_buf);                               /* errbuf */
```

# disperse.c, Command Line Argument Parsing

```
case 'b':
    /* destination MAC address */
    dp->bssid = libnet_hex_aton(optarg, &unused);
    break;
case 'D':
    /* destination IP address */
    if (!(dp->dst_ip = libnet_name2addr4(dp->l, optarg,
            LIBNET_RESOLVE)))
    {
        fprintf(stderr, "Bad destination IP address: %s\n",
                optarg);
        return (EXIT_FAILURE);
    }
    break;
case 'd':
    /* destination MAC address */
    dp->dst_mac = libnet_hex_aton(optarg, &unused);
    break;
case 'S':
    /* source IP address */
    if (!(dp->src_ip = libnet_name2addr4(dp->l, optarg,
            LIBNET_RESOLVE)))
    {
        fprintf(stderr, "Bad source IP address: %s\n", optarg);
        return (EXIT_FAILURE);
    }
    break;
case 's':
    /* source MAC address */
    dp->src_mac = libnet_hex_aton(optarg, &unused);
    break;
```

# disperse.c, Radiate Initialization

```
if (dp->src_mac == NULL || dp->dst_mac == NULL || dp->bssid == NULL ||
    dp->src_ip  == 0 || dp->dst_ip  == 0)
{
    usage(argv[0]);
    return (EXIT_FAILURE);
}

/* get our radiate context */
dp->r = radiate_init(0, err_buf);
if (dp->r == NULL)
{
    fprintf(stderr, "radiate_init(): %s", err_buf);
    return (EXIT_FAILURE);
}
```

# disperse.c, ICMP ECHO Packet Construction

```
ptag = libnet_build_icmpv4_echo(
    ICMP_ECHO,                                  /* type */
    0,                                          /* code */
    0,                                          /* checksum */
    0x0600,                                     /* id */
    0x0e00,                                     /* sequence number */
    NULL,                                       /* payload */
    0,                                          /* payload size */
    dp->l,                                      /* libnet context */
    0);
if (ptag == -1)
{
    fprintf(stderr, "Can't build ICMP header: %s\n",
            libnet_geterror(dp->l));
    return (EXIT_FAILURE);
}
```

# disperse.c, IP Packet Construction

```
ptag = libnet_build_ipv4(
    LIBNET_IPV4_H + LIBNET_ICMPV4_ECHO_H,       /* length */
    0,                                          /* TOS */
    0xfe,                                       /* IP ID */
    0,                                          /* IP Frag */
    128,                                        /* TTL */
    IPPROTO_ICMP,                               /* protocol */
    0,                                          /* checksum */
    dp->src_ip,                                 /* source IP */
    dp->dst_ip,                                 /* destination IP */
    NULL,                                       /* payload */
    0,                                          /* payload size */
    dp->l,                                      /* libnet context */
    0);
if (ptag == -1)
{
    fprintf(stderr, "Can't build IP header: %s\n", libnet_geterror(dp->l));
    return (EXIT_FAILURE);
}

if (libnet_toggle_checksum(dp->l, ptag, LIBNET_ON) == -1)
{
    fprintf(stderr, "Can't toggle checksum flag on for IP header %s\n",
            libnet_geterror(dp->l));
    return (EXIT_FAILURE);
}
```

# disperse.c, 802.11 Data Frame Creation

```
ip_packet   = NULL;
ip_packet_s = 0;
libnet_adv_cull_packet(dp->l, &ip_packet, &ip_packet_s);

/* build the payload */
payload_s = LIBNET_IPV4_H + LIBNET_ICMPV4_ECHO_H + LIBNET_802_2SNAP_H;
payload   = malloc(payload_s);
if (payload == NULL)
{
    fprintf(stderr, "malloc() %s\n", strerror(errno));
    return (EXIT_FAILURE);
}

memcpy(payload, llc_snap, LIBNET_802_2SNAP_H);
memcpy(payload + LIBNET_802_2SNAP_H, ip_packet, ip_packet_s);

dp->frame = radiate_build_data_frame(
        dp->bssid,                          /* BSSID */
        dp->dst_mac,                        /* destination MAC */
        dp->src_mac,                        /* source MAC */
        RADIATE_DATA_STYPE_DATA,            /* frame subtype */
        RADIATE_CTRL_TODS,                  /* frame control */
        payload,                            /* payload */
        payload_s,                          /* payload size */
        dp->r);                             /* radiate context */
```

# disperse.c, 802.11 Frame Injection

```
c = radiate_write(dp->frame, sizeof (struct hfa384x_tx_frame) +
        LIBNET_802_2SNAP_H + ip_packet_s, dp->r);
if (c == -1)
{
    fprintf(stderr, "radiate_write(): %s", radiate_geterror(dp->r));
}
else
{
    fprintf(stderr, "radiate_write(): wrote %d bytes\n", c);
}
free(dp->frame);

/* reset the card, which works half the time */
do_ioctl("0", dp->r);
do_ioctl("1", dp->r);
radiate_destroy(dp->r);
```

# Typical Radiate Program Usage

1. Make sure the library is built and installed. Duh.
2. Pop your Prism-2 card in… Wait for the DooDeet.
   - If you don't hear a DooDeet there is a driver / PCMCIA issue. But you're very smart so figuring it out will be a snap!
3. Put the card into monitor mode:
   - `mkultra:~/Libradiate-0.2/scripts# ./set_monitor 1`
4. Set the channel you want to do your work on:
   - `mkultra:~/Libradiate-0.2/scripts# ./set_channel 11`
5. Do it up!
   - `mkultra:~/Code/802.11/Tools/Omerta# ./omerta`

# The Warez

- You don’t get Omerta
- You do get Radiate (along with disperse):
  - http://www.packetfactory.net/Radiate

# 13db Gain Directional; Fun Stuff

# Known Issues

- We're at the mercy of the firmware…
- One call to write often results in several frames
- Problems with sending certain frames
  - To send data frames with the `TO_DS` bit set requires the module to be recompiled with `-DPRISM2_MONITOR_PACKET_INJECT`
    - Can't sniff frames when driver is built in this mode
  - Would be nice if we could make the card act as a passive radio
- Driver tends to crash or fail when card is removed and reseated frequently

# Futures

- This is pretty rough – we need a lot of code cleanup
  - Buffer management (pblocks?)
  - Possible API change to accommodate different layer 1 interfaces
- We want to support additional cards
  - Current issues with Prism-2 cards might go away
- Did someone say *Libnet Merger*? Could be!

# Summary

- The need exists to be able to develop arbitrary tools to test 802.11 networks for anomalous events
  - Byzantine failure was used as the example
- Radiate is a toolkit that allows the application programmer to develop tools to test for 802.11 security issues

- *Extra Special thanks to Timothy "The Newsh" Newsham*

# Building Open Source Network Security Tools

- Simple C library; A “component”
  - Upon which techniques are built
    - From which tools are created
- New book on how to rapidly develop your own network security tools
- New paradigm for describing a network security tool; accelerates conceptualization and development
- Wiley & Sons
- Due out in October 2002

Questions?

mike@stake.com