Data Exfiltration Using Covert Communication Channels

By Jake Valletta

June 8th, 2011

About Me

- Education
  - A. S. Hudson Valley Comm. College, 2009
  - B. S. Rochester Institute of Technology, 2011
    - Information Security & Forensics
- Experiences
  - Numerous Internships
  - MANDIANT Corp., June 2011
    - Pen testing / Incident Response
- Interests
  - Network Security & Forensics
  - Binary / Malware Analysis
  - Programming: C / Python

Agenda

- Data Exfiltration
- Covert Channel Basics
- Examples
- Demonstrations
- Detection Methods
- Conclusion

Data Exfiltration

Data Exfiltration

- The leaking of sensitive information
  - Company secrets
  - Source code
  - Client information
- A primary goal of an attacker
- Can have a big impact on company

Impact

- Loss of company & client information
- Company’s reputation at stake
  - Sony anyone…?
- Per state law, incidents must be reported in several states
  - *NYS Information Security Breach and Notification Act 2005*

Attack Life Cycle

- Reconnaissance
  - *Whois*, Company Website
- Scanning
  - Port scanning, service enumeration
- Gaining Access
  - Exploiting software, buffer overflows
- Maintaining Access
  - Root-kits, backdoors
- Covering Tracks & Hiding
  - Cleanse logs, *exfiltrate data*

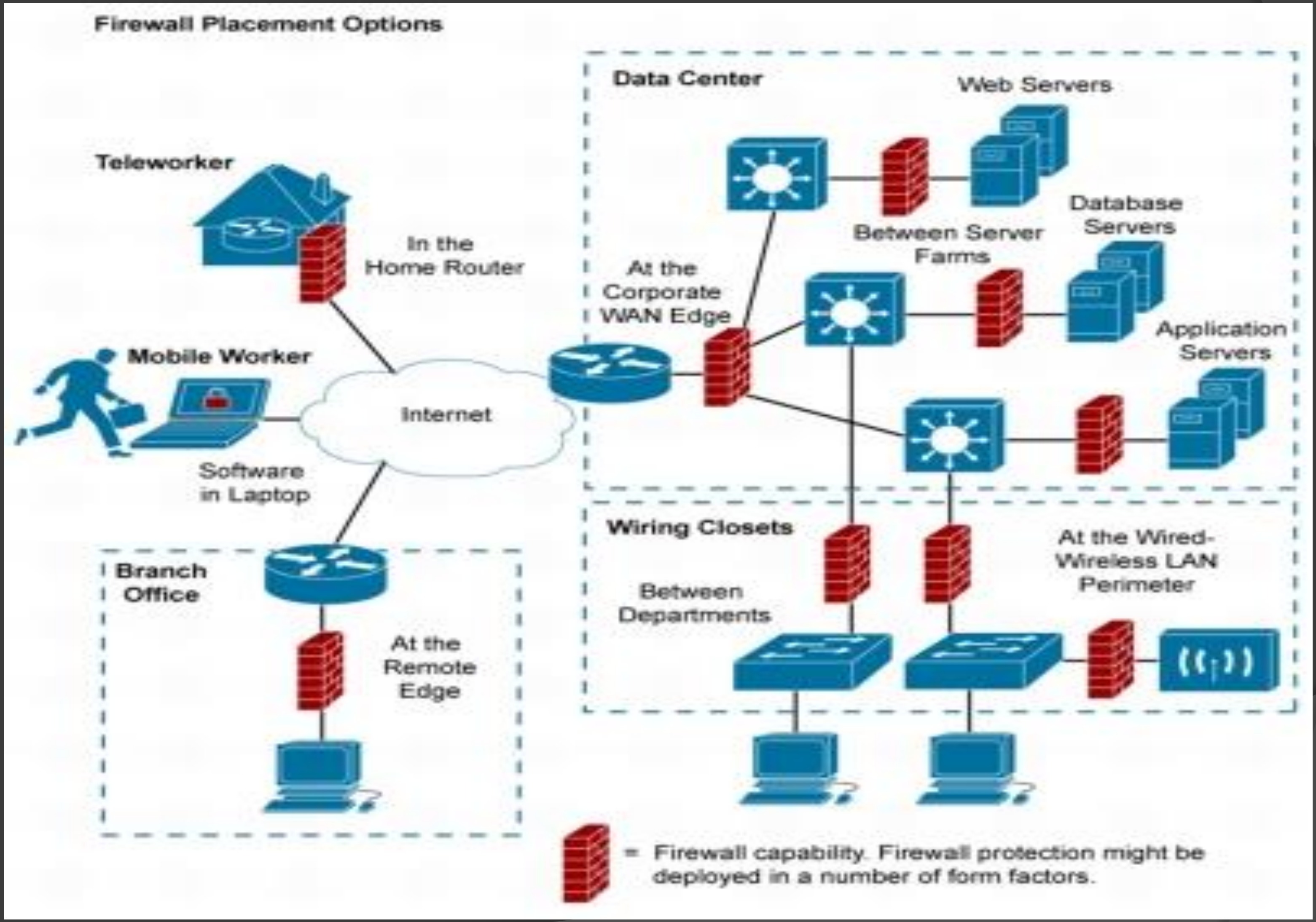
Source: Ed Skoudis, Tom Liston - Counter Hack Reloaded, 2006 (Pearson)

Exfiltration Methods

- Physical
  - USB Thief
  - Laptop Thief
- Cognitive
  - Social Engineering
  - Shoulder Surfing
- Network Based
  - FTP / SSH / HTTP
  - *Network–based Covert Channels*

Source: A. Giani et al. - Data Exfiltration and Covert Channels

…But I have a firewall(s), right?



Source: http://www.cisco.com

Firewalls: Not the Cure-all!

- Not as much focus on outbound traffic
- Majority are signature-based
- Need to be configured properly to be effective

Covert Channel Basics

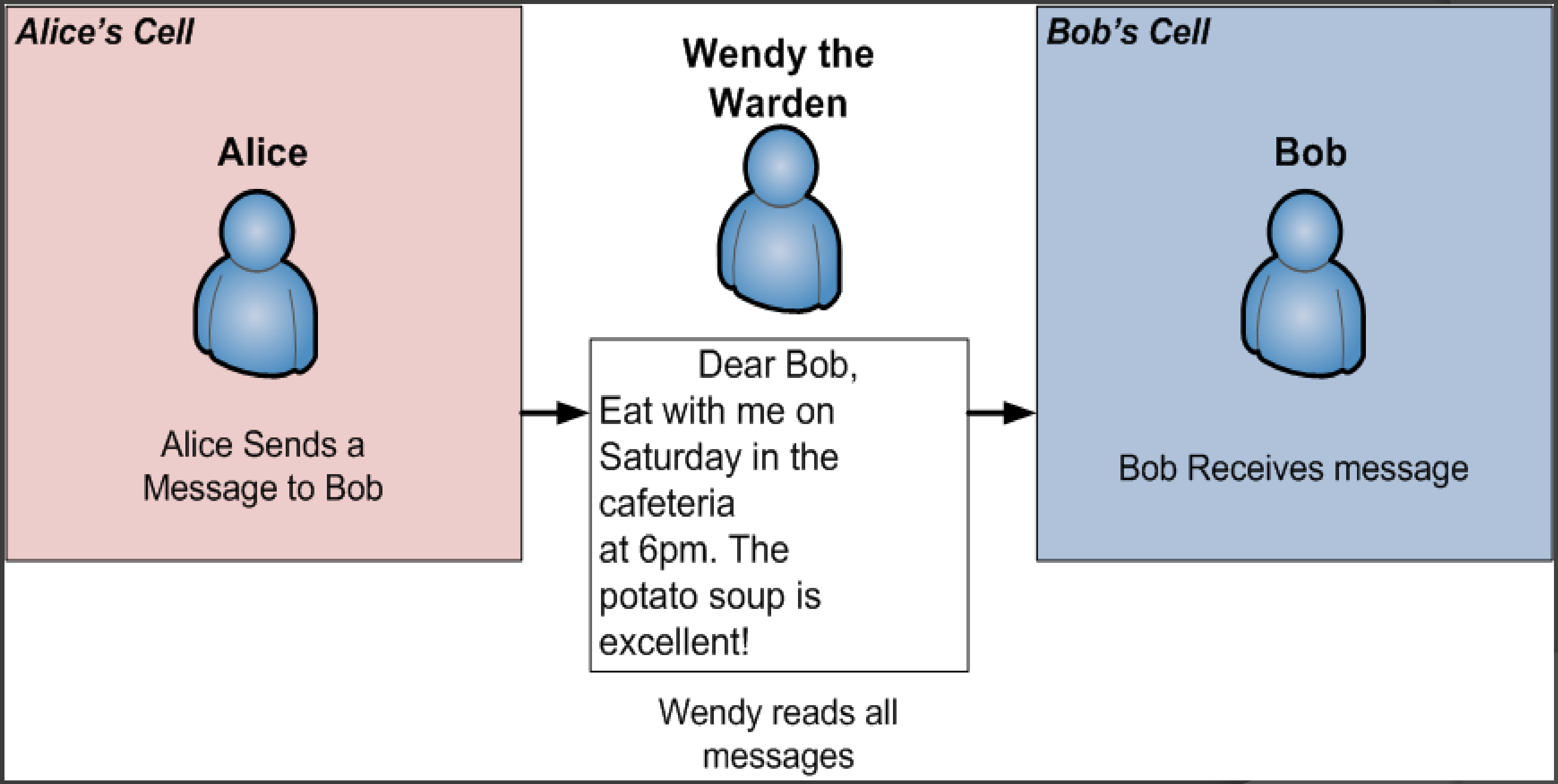
Covert Channels

"Covert channels use means of communication not normally intended to be used for communication, making them quite elusive."

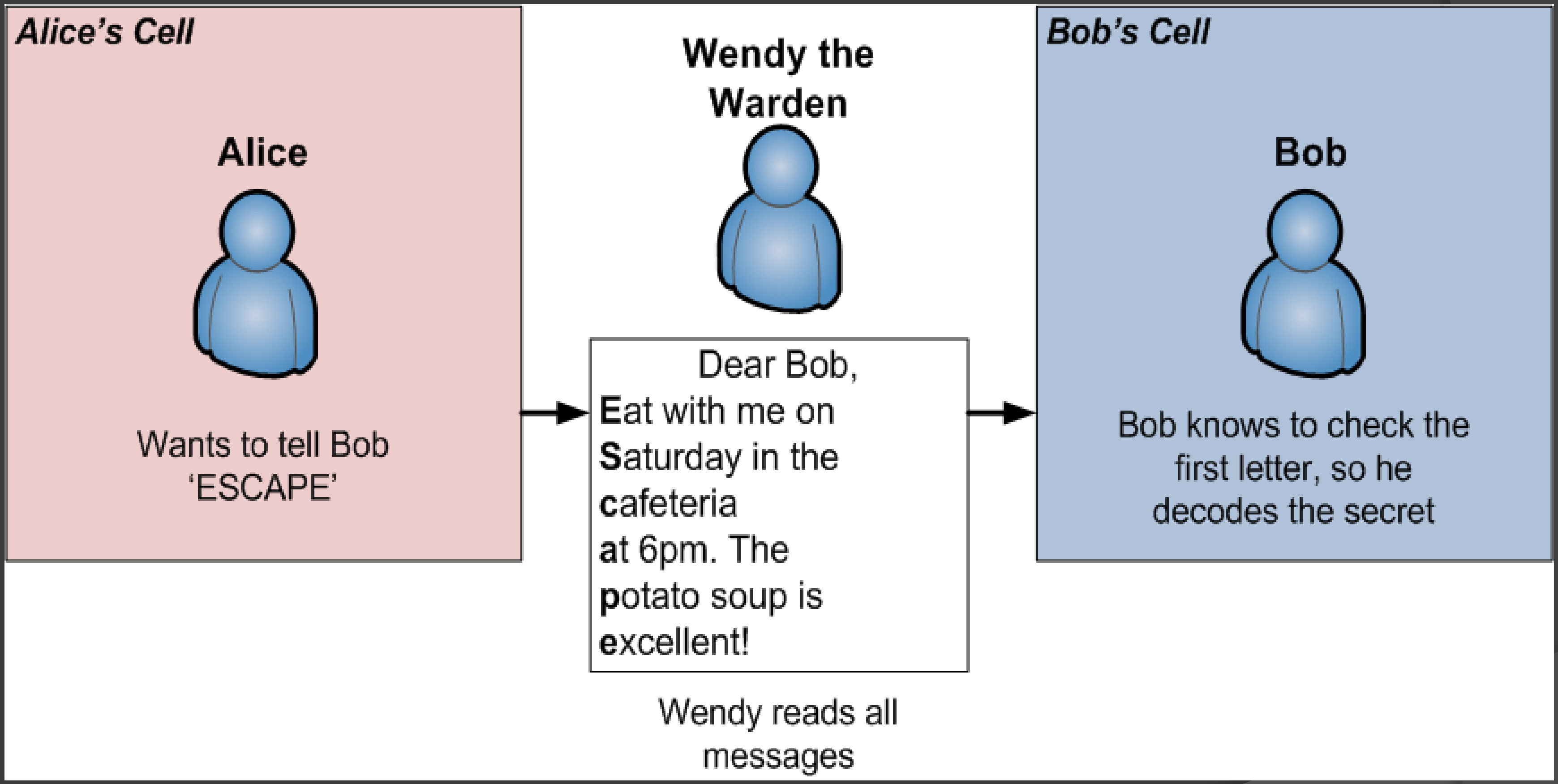
"Encryption only protects communication from being decoded by unauthorized parties, whereas covert channels aim to hide the very existence of the communication."

Source: caia.swin.edu.au/cv/szander/publications/szander-ieee-comst07.pdf

Prisoner Problem



Prisoner Problem



Prisoner Problem

- Allows a secret communication channel across an unsecure channel
- Nothing unordinary is observed, so it is stealthy
- Role of Wendy the Warden can impact the channel's effectiveness
  - Active, Passive, Malicious

Covert Channel Types

- Storage Based
  - The information we want to send is ‘stored’ somewhere in the overt communication channel
- Timing Based
  - The timing of an overt communication channel is the covert channel

Storage Channels

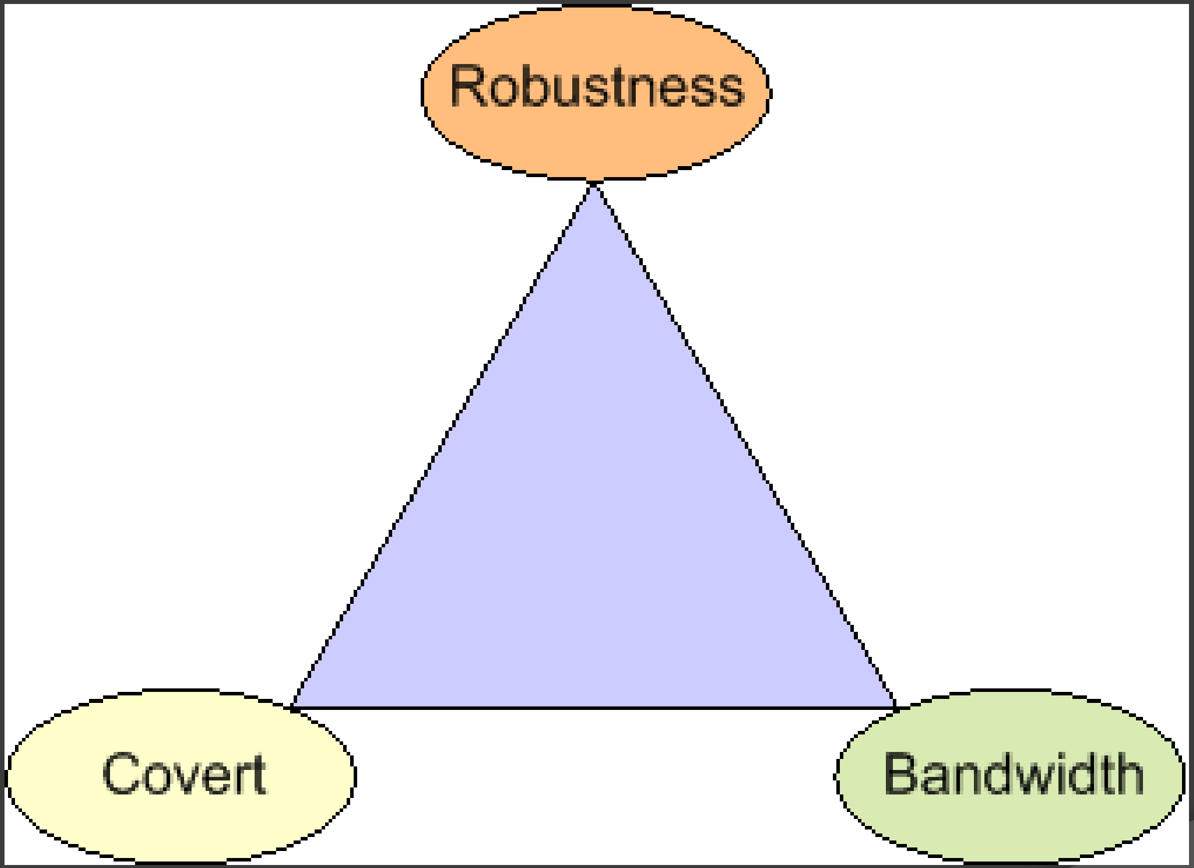
- Hide data in protocol headers
- Requires modification of overt channel, OR a ‘fake’ overt channel
- Some can be detected and mitigated with proper firewall rules

Timing Channels

- Very difficult to create
  - Latency issues
- Very difficult to find
- Doesn’t require modification to an existing communication stream

Building Covert Channels

Things to Consider



Things to Consider

- Do we need bidirectional or unidirectional covert channel?
- What kind of warden is present?

Python

- Modern interpreted programming language
  - Powerful, fast & easy to follow syntax
  - Extensive built-in libraries
  - Plays well with C / Java / .NET code
- Open-source
- Language of choice for ‘hackers’ and reverse-engineers
- Excellent for prototyping and POC code

Python Website: http://www.python.org/

Scapy

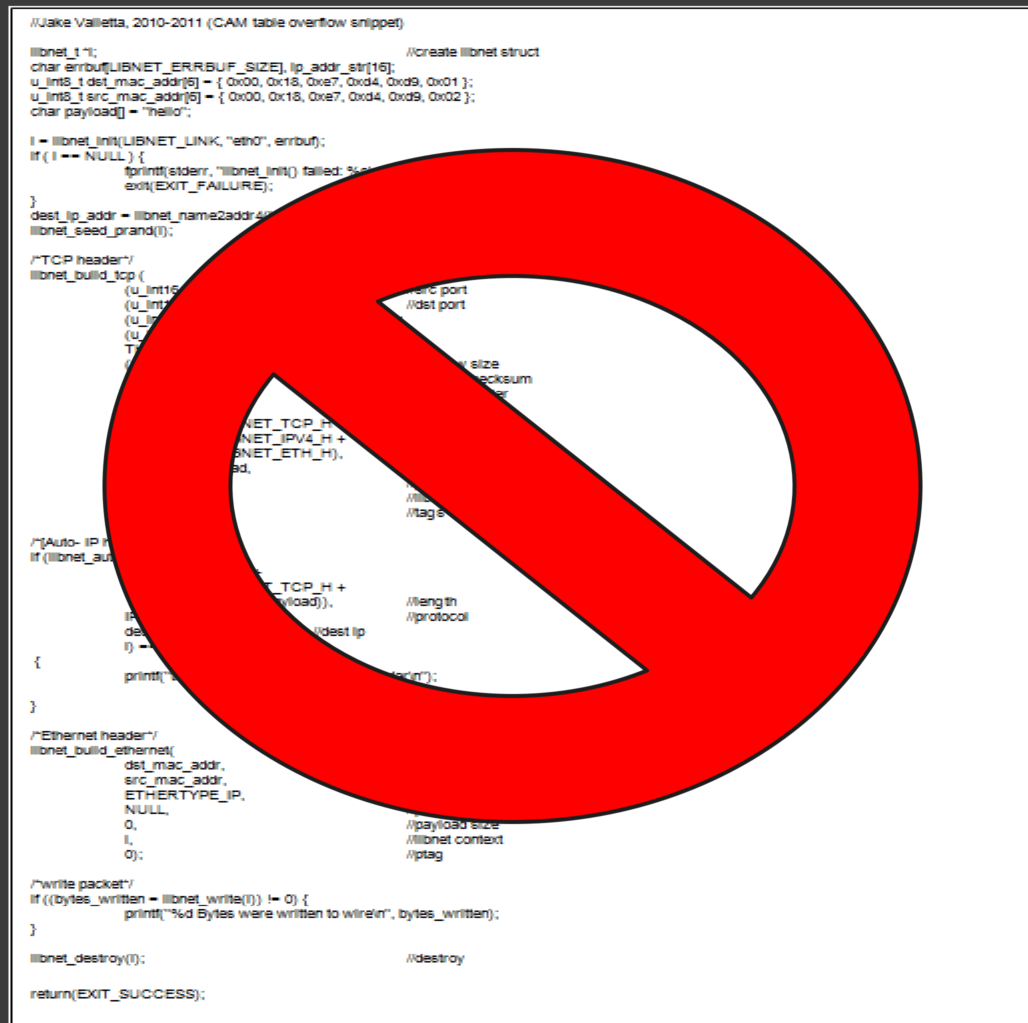
- Powerful interactive packet manipulation program
  - Forge and decode custom packets
  - Sniff network traffic or read captured packets
  - Combines functionality of many tools
    - *nmap*, *hping3*, *p0f*, *tcpdump*
- Can import into Python 2.5+

Scapy Website: http://www.secdev.org/projects/scapy/

Coding a TCP Packet in C

```
//Jake Valletta, 2010-2011 (CAM table overflow snippet)

libnet_t *l;                                              //create libnet struct
char errbuf[LIBNET_ERRBUF_SIZE], ip_addr_str[16];
u_int8_t dst_mac_addr[6] = { 0x00, 0x18, 0xe7, 0xd4, 0xd9, 0x01 };
u_int8_t src_mac_addr[6] = { 0x00, 0x18, 0xe7, 0xd4, 0xd9, 0x02 };
char payload[] = "hello";

l = libnet_init(LIBNET_LINK, "eth0", errbuf);
if ( l == NULL ) {
        fprintf(stderr, "libnet_init() failed: %s\n", errbuf);
        exit(EXIT_FAILURE);
}
dest_ip_addr = libnet_name2addr4(l, "192.168.000.003", LIBNET_DONT_RESOLVE);
libnet_seed_prand(l);                         //seed random

/*TCP header*/
libnet_build_tcp (
        (u_int16_t)999,                                   //src port
        (u_int16_t)999,                                   //dst port
        (u_int32_t)1,                 //seq number
        (u_int32_t)1,                 //ack number
        TH_PSH,                       //PSH flag
        (u_int16_t)10000,                                 //window size
        0,                                                //autofill checksum
        0,                                                //urgent pointer
        (sizeof(payload) +                                //size of packet
                LIBNET_TCP_H +
                LIBNET_IPV4_H +
                LIBNET_ETH_H),
        (u_int8_t*)payload,                               //payload
        sizeof(payload),                                  //payload size
        l,                                                //libnet context
        0);                                               //tags

/*[Auto- IP header*/
if (libnet_autobuild_ipv4 (
         (LIBNET_IPV4_H +
                LIBNET_TCP_H +
                sizeof(payload)),                         //length
        IPPROTO_TCP,                                      //protocol
        dest_ip_addr,                 //dest ip
        l) == -1 )
 {
        printf("there was a problem with IP header\n");
                exit(EXIT_FAILURE);
}

/*Ethernet header*/
libnet_build_ethernet(
        dst_mac_addr,                                     //dest
        src_mac_addr,                                     //source
        ETHERTYPE_IP,                                     //type
        NULL,                                             //payload
        0,                                                //payload size
        l,                                                //libnet context
        0);                                               //ptag

/*write packet*/
if ((bytes_written = libnet_write(l)) != 0) {
        printf("%d Bytes were written to wire\n", bytes_written);
}

libnet_destroy(l);                                        //destroy

return(EXIT_SUCCESS);
```

Coding a TCP Packet in C



```
//Jake Valletta, 2010-2011 (CAM table overflow snippet)

libnet_t *l;                                              //create libnet struct
char errbuf[LIBNET_ERRBUF_SIZE], ip_addr_str[16];
u_int8_t dst_mac_addr[6] = { 0x00, 0x18, 0xe7, 0xd4, 0xd9, 0x01 };
u_int8_t src_mac_addr[6] = { 0x00, 0x18, 0xe7, 0xd4, 0xd9, 0x02 };
char payload[] = "hello";

l = libnet_init(LIBNET_LINK, "eth0", errbuf);
if ( l == NULL ) {
        fprintf(stderr, "libnet_init() failed: %s
        exit(EXIT_FAILURE);
}
dest_ip_addr = libnet_name2addr4(
libnet_seed_prand(l);

/*TCP header*/
libnet_build_tcp (
        (u_int16                                          //src port
        (u_int1                                           //dst port
        (u_i
        (u_
        T
        (                                                 size
                                                          ecksum

                  NET_TCP_H
                  NET_IPV4_H +
                  BNET_ETH_H),
                  ad,

                                                          //li
                                                          //tag s

/*[Auto- IP h
if (libnet_au

                  T_TCP_H +
                  yload)),                                //length
        IP                                                //protocol
        de                             //dest ip
        l) ==
 {
        printf("                                          \n");

}

/*Ethernet header*/
libnet_build_ethernet(
        dst_mac_addr,
        src_mac_addr,
        ETHERTYPE_IP,
        NULL,
        0,                                                //payload size
        l,                                                //libnet context
        0);                                               //ptag

/*write packet*/
if ((bytes_written = libnet_write(l)) != 0) {
        printf("%d Bytes were written to wire\n", bytes_written);
}

libnet_destroy(l);                                        //destroy

return(EXIT_SUCCESS);
```

…And with Scapy

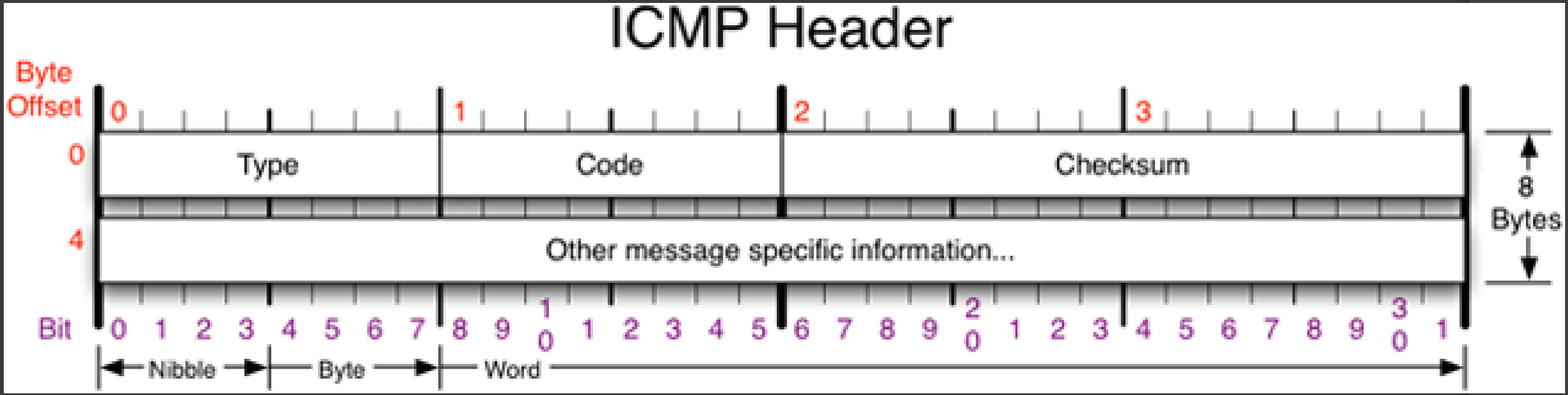
```
root@bt:~# scapy
Welcome to Scapy (2.1.0)
>>> ether_layer = Ether(src='00:11:22:33:44:55', dst='55:44:33:22:11')
>>> ip_layer = IP(src='192.168.0.100', dst='192.168.0.101')
>>> tcp_layer = TCP(seq=0x1234, dport=999L, sport=999L, flags=8)
>>> sendp(ether_layer/ip_layer/tcp_layer/'hello')
.
Sent 1 packets.
```

Example #1 - ICMP

ICMP – The Protocol

- Internet Control Message Protocol
- Used in error reporting & network diagnostics
  - ‘ping’ (Echo Request / Reply)
  - Windows ‘tracert’ (TTL Exceeded)
  - Need to Fragment, Destination Unreachable, Port Administratively Filtered, Redirect, etc.
- Should be disabled (?)

The ICMP Header



Source: http://www.insecure.in/packet_header_analysis.asp

ICMP Echo Request

```
⊞ Frame 539 (74 bytes on wire, 74 bytes captured)
⊞ Ethernet II, Src: AlpsElec_6d:df:74 (00:1e:3d:6d:df:74), Dst: All-HSRP-routers_01 (00:00:0c:07:ac:01)
⊞ Internet Protocol, Src: 129.21.62.207 (129.21.62.207), Dst: 74.125.226.148 (74.125.226.148)
⊟ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0 ()
    Checksum: 0x4d5a [correct]
    Identifier: 0x0001
    Sequence number: 1 (0x0001)
  ⊟ Data (32 bytes)
      Data: 6162636465666768696A6B6C6D6E6F7071727374757677 61...
      [Length: 32]

0000  00 00 0c 07 ac 01 00 1e  3d 6d df 74 08 00 45 00   ........ =m.t..E.
0010  00 3c 07 8a 00 00 80 01  46 41 81 15 3e cf 4a 7d   .<...... FA..>.J}
0020  e2 94 08 00 4d 5a 00 01  00 01 61 62 63 64 65 66   ....MZ.. ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi
```

**ICMP Echo created by Windows NT TCP/IP Stack*

Analysis

- Type – 0x08 (Echo Request)
  - Distinguishes this as a ‘ping’
- Code – 0x00
- Checksum
  - Checked for packet integrity by routers
- ID – 0x0001
- Sequence – 0x0001
- Data – 32 bytes
  - Of what…?

Exploring

- According to RFC 792, the only value for the code field in an ICMP Echo message is 0.
  - Code is used in other ICMP messages (think ‘subtype’)
  - Changing the code does not invalidate the message
- ID differentiates sessions, much like a TCP / UDP port
  - Changing the ID does not invalidate the message
- Sequence is a counter for a session
  - Changing the Sequence does not invalidate the message

Options

- A storage based covert channel can be created using these fields
  - Each field can hold data to be sent
- Data can be tunneled over the payload field
  - Encryption to obscure context
- Shouldn’t be detected / blocked by IDS or Firewall

Restrictions

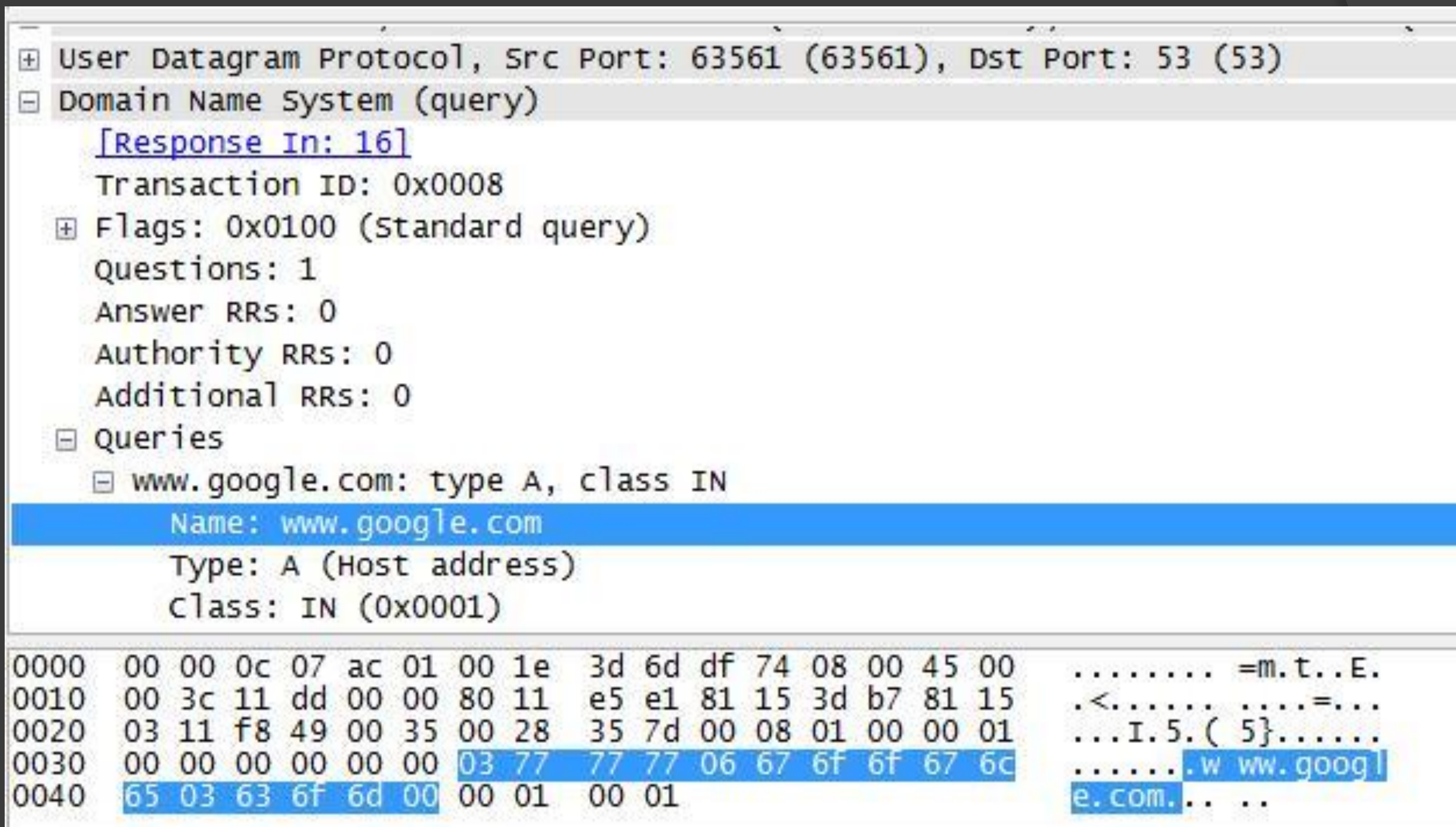
- Some networks filter / drop ICMP traffic
  - Superfluous traffic
  - Additional attack vector
- Could be detected by IDS
  - Why so many pings?
- Concept has been around for awhile
  - *lokid* (Phrack Magazine, 1997)

Example #2 - DNS

DNS – The Protocol

- Used primarily for name resolution
  - *What is the IP address for www.google.com?*
- Hierarchical design
- Must be allowed in and out of firewall

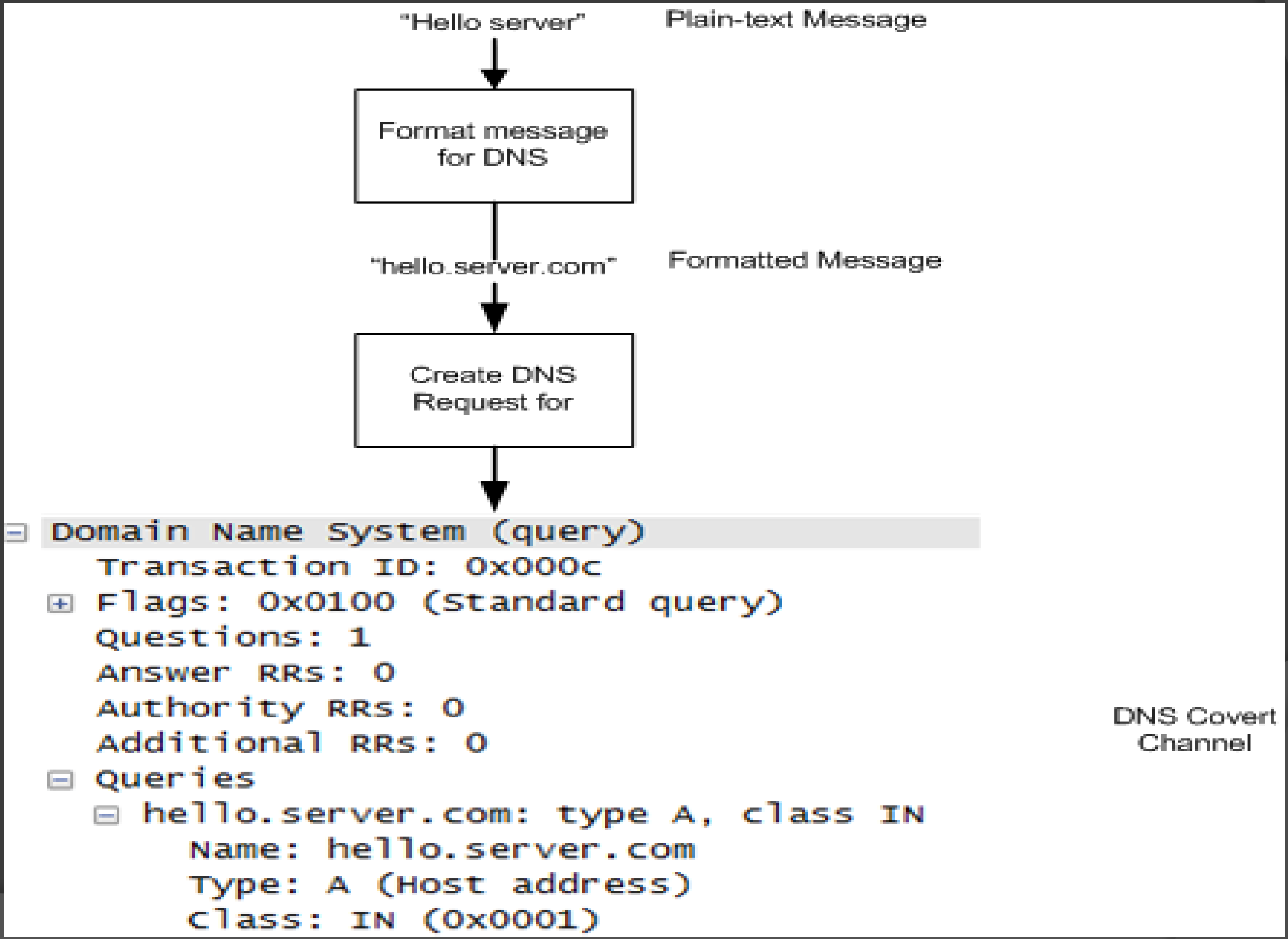
A DNS Request



Exploring

- The query of the request could be modified
  - DNS lookups for A, CNAME and TXT records
  - The ‘Name’ field can contain our data
- Multiple ‘questions’ can be specified
  - But packet size must be less than MTU, as DNS sets ‘Don’t Fragment’ flag in IP header (per RFC)
- Valid DNS requests use character: [a-zA-Z0-9\-]

Example Flow



Options

- Looks like a legitimate DNS request
  - How can an IDS tell it’s forged?
- Encryption can obscure the message
- Provides a good **unidirectional** covert channel
  - Can be made bidirectional with CNAME / TXT requests (*OZYmanDNS, NSTX*)

Advantages

- Shouldn’t be blocked by any firewall
  - DNS is required to be allowed out of the firewall
- Very hard to detect or filter
  - You’d be surprised what domains exist
- Even if it is detected, encryption can protect payload

Example #3 – ICMPv6

IPv6 / ICMPv6 – The Protocols

- Next Generation
  - Development started in early 1990s
- Secure (?)
- Slowly but surely replacing IPv4
- ICMPv6 is integrated into IPv6
  - Neighbor Discovery Protocol (NDP)

ICMPv6 Echo Request

```
⊟ Internet Protocol Version 6, Src: fe80::20c:29ff:fe2c:3675 (fe80::20c:29ff:fe2c:3675),
  ⊞ 0110 .... = Version: 6
  ⊟ .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
      .... 0000 00.. .... .... .... .... .... = Differentiated Services Field: Default
      .... .... ..0. .... .... .... .... .... = ECN-Capable Transport (ECT): Not set
      .... .... ...0 .... .... .... .... .... = ECN-CE: Not set
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 40
    Next header: ICMPv6 (0x3a)
    Hop limit: 128
    Source: fe80::20c:29ff:fe2c:3675 (fe80::20c:29ff:fe2c:3675)
    [Source SA MAC: Vmware_2c:36:75 (00:0c:29:2c:36:75)]
    Destination: fe80::20c:29ff:feca:9906 (fe80::20c:29ff:feca:9906)
    [Destination SA MAC: Vmware_ca:99:06 (00:0c:29:ca:99:06)]
⊟ Internet Control Message Protocol v6
    Type: 128 (Echo (ping) request)
    Code: 0 (Should always be zero)
    Checksum: 0xb356 [correct]
    ID: 0x0000
    Sequence: 23
  ⊟ Data (32 bytes)
      Data: 6162636465666768696a6b6c6d6e6f7071727374757677761...
      [Length: 32]
```

Exploring

- Traffic Class is the replacement for ‘Type of Service’ in IPv4
  - Used in real-time data (VoIP)
- Flow Label is used to quickly process real-time data
  - Saves time by not examining entire header, because it already knows about this ‘flow’
- Code, Sequence, and ID are still the same
  - Ping6ed machine won’t respond if code isn’t 0

Options

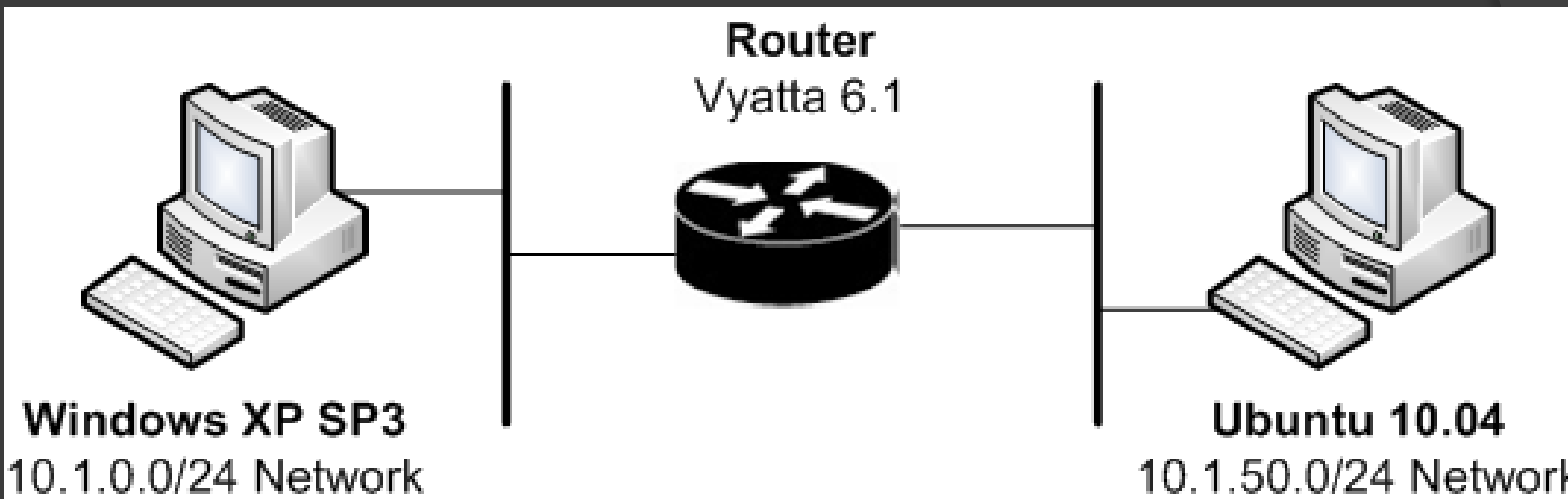
- Traffic Class & Flow Label can be modified
  - Shouldn’t affect packets travel (?)
- Modulate ICMPv6 fields
  - Just like ICMPv4
- Tunnel Data in payload section
  - *v00d00N3t* (R. P. Murphy, DEFCON14)

Advantages

- Still not fully understood / deployed
  - Firewalls / IDS might not be fully aware
  - RFC's might not be strictly followed
- ICMPv6 cannot be turned off anymore
  - "*ICMPv6 is an integral part of IPv6 and MUST be fully implemented by every IPv6 node.*" (RFC 2463)

Demonstration

Topology


Router
Vyatta 6.1
Windows XP SP3
10.1.0.0/24 Network
Ubuntu 10.04
10.1.50.0/24 Network

Detection Methods?

(Good luck!)

The Problem

- The very nature of a covert channel makes it hard to find
  - How do you know to look for something that you don't know you needed to look for?
- Once you do detect it, how do you stop it?
  - The data is already leaked!

Solutions

- Signature-based Approach
  - How most antivirus, DLP, IDS & IPS solutions work
  - Will not detect new covert channels
  - Resource intensive
- Behavioral-based Approach
  - Not as common
  - Resource intensive (full packet inspection)
  - Capability to detect known and unknown storage channels

Questions and Comments?

Contact Information

- Email
  - javallet@gmail.com
  - jrv1197@rit.edu
- LinkedIn
  - http://www.linkedin.com/pub/jacob-valletta/20/aa1/57
- Questions, ideas, source-code, projects, etc.