Steganography and Erlang

Derek Brown
derekbrown121@gmail.com

Overview

Steganography – Concealing a message or file within another file or communication medium

Agenda

- General concepts
- Discuss 2 types:
  - Within images
  - Within ICMP Echo packets
- Code samples and demos

Examples

Examples of steganographic techniques:

- Low-order bits of image or sound files
- Fields in network packets
- Echo in sound files
- Whitespace manipulation
- XML: <img></img> vs. <img/>

Uses

- Digital watermarking
- Communication when communication is monitored or restricted
- Fun

Steganalysis

Detecting messages hidden with steganographic techniques

Comparing with source files from the same source (e.g., a camera)

Statistical analysis of characteristics of the data

Basic Strategy for Images – First Pass

Embedding Message

- The bits of the source bytes will be embedded
- Set the low-order bit of each destination byte to next source bit

Basic Strategy for Images – First Pass

Embedding Message

- The bits of the source bytes will be embedded
- Set the low-order bit of each destination byte to next source bit

Extracting Message

- Pull out the low-order bit of each byte
- Build the output bytes from the bits

Using Evan Miller's erl_img for the image reading, writing, and image format handling

Message Extraction, Demo 1

```
1> img_png:read_and_write_message(
  ”innocuous.png", ”mystery.out").
```

Output file could be anything

Can use the “file” command to determine

```
$ file mystery.out
$ mv mystery.out mystery.png
```

When to Stop Extracting Bits?

Include 2-byte length before message

Thus first 16 bytes of output each contain 1 bit from the length

Limits message length to 65535 bytes

(2 ^ 16) – 1

What if the Message is Too Big?

What if the Message is Too Big?

Allow N low-order bits to be set

But, more bits set -> Easier to visually detect

Message Embedding, Demo 2

For a 300x300 image:

- 300 * 300 pixels = 90,000 pixels total
- 90,000 pixels * 3 bytes per pixel (RGB) = 270,000 bytes
- 270,000 bytes - 40 for our header (to be examined shortly) = 269,960 bytes available
- 269,960 div 8 == 33,745 max message size with 1 bit per byte

Message Embedding, Demo 2

```
1> img_png:add_message_from_file(
 ”demo2.dat", "all_black_300x300.png”,
 ”demo2.png").
```

Message Embedding, Demo 2

```
1> img_png:add_message_from_file(
  ”demo2.dat", "all_black_300x300.png”,
  ”demo2.png").
```

Fails because demo2.dat is too large. Solution:

```
2> img_png:add_message_from_file(
  ”demo2.dat", "all_black_300x300.png”,
  ”demo2.png”, [{bits_per_byte, 2}]).
```

Message Embedding, Demo 2

We can increase bits_per_byte and see how it visually affects the output

What if the top of the image isn’t noisy?

The top of the image may not be the noisiest part of it

What if the top of the image isn't noisy?

The top of the image may not be the noisiest part of it

Solution: specify an offset % to start the embedding

Avoiding dense modified sections

With high bits-per-byte, visible changes appear packed together

Avoiding dense modified sections

With high bits-per-byte, visible changes appear packed together

Solution: specify byte spacing

High-level Embedding Entrypoints

```
img_png:add_message_from_file(
  MsgFile, InFile, OutFile, Opts)

img_png:add_message(
  Msg, InFile, OutFile, Opts)
```

Embedding Initialization

```
{ok, Img} = erl_img:load(ImgPath),
Pixmap = hd(Img#erl_image.pixmaps),
Pixels = Pixmap#erl_pixmap.pixels,
[First | Rest] = Pixels,
State = #state{
  curr_row = First,
  rest_rows = Rest,
  img = Img}.
```

Embedding Execution

Each bit to set is destined for a specific byte and bit in the image

Create list of {ByteNum, BitNum, BitVal} specifications

Given that specification, just do it (read: apply the spec to the source file)

Header

Header values:

- Number of embedded source bytes: 16 bits
- Offset Percent: 8 bits
- Spacing: 8 bits
- Bits-per-byte: 8 bits

Header Specification Example

Specification for offset percentage of 5
Tuple members: {ByteNum, BitNum, BitVal}
[{16, 0, 1},
 {17, 0, 0},
 {18, 0, 1},
 {19, 0, 0},
 {20, 0, 0},
 {21, 0, 0},
 {22, 0, 0},
 {23, 0, 0}]

Message Specification

- Create a list of the bits from the message

Message Specification

- Create a list of the bits from the message
- Given the message length, offset, spacing, and bits-per-byte
  - Calculate a list of the destination byte num/bit num for each bit in the message

Message Specification

- Create a list of the bits from the message
- Given the message length, offset, spacing, and bits-per-byte
  - Calculate a list of the destination byte num/bit num for each bit in the message
- Combine the two lists to create a specification of what to set (as done for the header)

Message Specification Part 1 – Message Bits

1. Convert Byte to big-endian list of bits:

```
[X || <<X:1>> <= <<Byte:8>>]
```

Message Specification Part 1 – Message Bits

1. Convert Byte to big-endian list of bits:

```
[X || <<X:1>> <= <<Byte:8>>]
```

2. Convert each Byte in Message to a list of bits (list of lists)

```
BytesBigEndian = lists:map(
  fun(B) -> [X || <<X:1>> <= <<B:8>>] end,
  Message),
```

Message Specification Part 1 – Message Bits

3. Flip each list to little-endian

```
BytesLittleEndian = lists:map(
  fun(L) -> lists:reverse(L) end,
  BytesBigEndian),
```

Message Specification Part 1 – Message Bits

3. Flip each list to little-endian

```
BytesLittleEndian = lists:map(
  fun(L) -> lists:reverse(L) end,
  BytesBigEndian),
```

4. Flatten to one list

```
lists:flatten(BytesLittleEndian).
```

Message Specification Part 2 – Destination Byte/Bit Num

- Output: list of {ByteNum, BitNum}
- ByteNum:
  - Starts at Offset (converted from OffsetPercent)
  - Each is used BitsPerByte times
  - Advances by Spacing
- BitNum:
  - From 0 to BitsPerByte

Message Specification Part 2 – Destination Byte/Bit Num Example

First message byte, with Offset 2, Spacing 2, BitsPerByte 2:

[{42, 0},
 {42, 1},
 {44, 0},
 {44, 1},
 {46, 0},
 {46, 1},
 {48, 0},
 {48, 1},
 ...]

Message Specification: Combine Bits and Locations

Bits: [0, 0, 1, 0, ...]

Locations: [{42, 0}, {42, 1}, {44, 0}, {44, 1}, ...]

Combined: [{42, 0, 0}, {42, 1, 0}, {44, 0, 1}, {44, 1, 0}, ...]

Easy way to create Combined?

Message Specification: Combine Bits and Locations

Bits: [0, 0, 1, 0, ...]

Locations: [{42, 0}, {42, 1}, {44, 0}, {44, 1}, ...]

Combined: [{42, 0, 0}, {42, 1, 0}, {44, 0, 1}, {44, 1, 0}, ...]

```
lists:zipwith(
  fun({ByteNum, BitNum}, Bit)
      -> {ByteNum, BitNum, Bit} end,
  Locations,
  Bits).
```

Just Do It

Apply the message specification to the input pixels:

```
Pixels = Pixmap#erl_pixmap.pixels.
```

Contents of Pixels: `[{RowNum, BinaryBytes]}, ...]`

Example: `[{0, <<55,2,129,17>>}, ...]`

Really Just Do It

- For each tuple in specification, advance in the Pixels list to specified overall ByteNum

Really Just Do It

- For each tuple in specification, advance in the Pixels list to specified overall ByteNum
- Separate the byte on which to set a bit:

```
{Prefix, Byte, Postfix} =
  extract_parts_of_row(
    BinaryBytes, RowByteNum)
```

Really Just Do It

- For each tuple in specification, advance in the Pixels list to specified overall ByteNum
- Separate the byte on which to set a bit:

```
{Prefix, Byte, Postfix} =
  extract_parts_of_row(
    BinaryBytes, RowByteNum)
```

- Set bit BitNum of Byte to Bit

Really Just Do It

- For each tuple in specification, advance in the Pixels list to specified overall ByteNum
- Separate the byte on which to set a bit:

```
{Prefix, Byte, Postfix} =
  extract_parts_of_row(
    BinaryBytes, RowByteNum)
```

- Set bit BitNum of Byte to Bit
- Reconstruct the row binary

```
  <<Prefix/binary, ModifiedByte, Postfix/
binary>>
```

Message Extraction - Header

- Prepare specification of byte num/bit num pairs to read the header

  Example: OffsetPct is in bit 0 of bytes 8 though 15

```
[{8, 0}, {9, 0}, {10, 0}, {11, 0}, {12, 0},
 {13, 0}, {14, 0}, {15, 0}]
```

Message Extraction - Header

- Prepare specification of byte num/bit num pairs to read the header

  Example: OffsetPct is in bit 0 of bytes 8 though 15

```
[{8, 0}, {9, 0}, {10, 0}, {11, 0}, {12, 0},
 {13, 0}, {14, 0}, {15, 0}]
```

- Read the specified bits

Message Extraction - Header

- Prepare specification of byte num/bit num pairs to read the header

  Example: OffsetPct is in bit 0 of bytes 8 though 15

```
[{8, 0}, {9, 0}, {10, 0}, {11, 0}, {12, 0},
 {13, 0}, {14, 0}, {15, 0}]
```

- Read the specified bits
- Convert the bits to an integer (reversing them to be big-endian):

```
<<Int:8>> = << <<Bit:1>> ||
    Bit <- lists:reverse(Bits) >>
```

Message Extraction - Message

Given the message length, offset, spacing, and bits-per-byte from the header:

- Calculate a list of the source byte num/bit num for each bit in the message
- Read the specified bits
- Convert the bits into a list of bytes

Message Embedding, Demo 3

Vary spacing and offset

```
Options = [{bits_per_byte, 6}],
img_png:add_message_from_file(
  "demo3.dat", "building.png”, "demo3.png",
  Options).
```

Also try:

```
Options = [{bits_per_byte, 6}, {spacing, 6}].
Options = [{bits_per_byte, 6}, {offset_pct,
30}].
```

Embedding in ICMP Echo Request Packets

ICMP: Internet Control Message Protocol

Sits on top/part of IP packets

This technique motivated by an article in First Monday by Craig Rowland

Embedding in ICMP Echo Request Packets

ICMP used by network devices for communicating status or other information

- Packet destination unreachable
- Redirect to alternate router
- Test reachability (echo request/reply (used to ping))

ICMP Echo Request Header

Type: 8 bits
- Echo Request: 8

Code: 8 bits
- Must be 0

Header Checksum: 16 bits

Identifier: 16 bits
- Arbitrary

Sequence Number: 16 bits
- Arbitrary. Usually increasing number.

Data (payload)
- Arbitrary, but risks fragmentation if too large

ICMP Echo Request / Ping Demo

Can use tools such as tcpdump or Wireshark to examine packets in transit

```
$ sudo tcpdump -X -i lo0 \
  'icmp[icmptype] == icmp-echo'

$ ping -c1 -pff 127.0.0.1
```

ICMP Echo Request Embed Location

An easy place to insert a message in an ICMP packet?

ICMP Echo Request Embed Location – Option 1

The payload field!

Demo:

```
icmp_data_receive_server:start_receive(
    payload).
icmp_data_send:send(
    "127.0.0.1", payload, “Shhh!”).
```

ICMP Echo Scheme– Payload

Problem: it's visible via tcpdump, etc.

We'll look at other options.

gen_icmp

Abusing gen_udp

for great good!

Sending Echo Packets with gen_icmp

A project from Michael Santos

Allows sending of manually-constructed ICMP packets (not just echo)

Sending Echo Packets with gen_icmp

```
{ok, Socket} = gen_icmp:open().

Packet = gen_icmp:echo(
  Family, Id, SequenceNumber,
  Payload).

gen_icmp:send(Socket, Host, Packet).
```

Loop if more data

Receiving Echo Packets with gen_icmp

```
{ok, Socket} = gen_icmp:open().

{ok, {Address, Packet}} =
  gen_icmp:recv(Socket, Length, Timeout).
```

Loop

ICMP Echo Scheme - Payload

Send

For each block of size N of the message:

- Id: random for given data to send
- Sequence number: increasing from 0
- Payload: the block

ICMP Echo Scheme - Payload

Send

For each block of size N of the message:

- Id: random for given data to send
- Sequence number: increasing from 0
- Payload: the block

Receive

```
[_Ipv4Hdr, _IcmpHdr, Payload] =
    pkt:decapsulate(ipv4, Packet),
erlang:binary_to_list(Payload).
```

ICMP Echo Scheme - Payload

It’s too visible!

Other options?

ICMP Echo Request Embed Location – Option 2

The identifier field!

ICMP Echo Scheme – Identifier

Send

For each 2 bytes of the message (or 1 byte at end):

- Id: `(M bsl 8) bor N` (or just `(M bsl 8)` if last single byte to send)
- Sequence number: 0
- Payload: random

ICMP Echo Scheme – Identifier

Send

For each 2 bytes of the message (or 1 byte at end):

- Id: (M bsl 8) bor N (or just (M bsl 8) if last single byte to send)
- Sequence number: 0
- Payload: random

Receive

```
[_Ipv4Hdr, IcmpHdr, _Payload] =
    pkt:decapsulate(ipv4, Packet),
<<M:8, N:8>> = <<IcmpHdr#icmp.Id:16>>,
[M, N].
```

ICMP Echo Scheme– Identifier

Demo:

```
icmp_data_receive_server:start_receive(
    identifier).
icmp_data_send:send(
    "127.0.0.1", identifier, “Shh!”).
```

ICMP Echo Scheme– Identifier

Problem: it's visible via tcpdump, etc. (just less obvious than payload)

Other options?

Can we communicate data without directly sending the bits/bytes of the message?

ICMP Echo Request Embed Location – Option 3

Use the parity of the length of the (arbitrary) payload represent a bit of the message!

Payload length is even: 0

Payload length is odd: 1

Thus 8 packets to send each byte

ICMP Echo Scheme– Payload Length

Problems using multiple packets to send each byte (apart from being slow)?

ICMP Echo Scheme– Payload Length

Problems using multiple packets to send each byte (apart from being slow)?

Sending IP packets is unreliable (unlike TCP)

A dropped packet would skew all those after it

We need to know when we're at a byte boundary

ICMP Echo Scheme– Payload Length

Send

For each bit of the message:

- Id: random for given message to send
- Sequence number: increasing from 0
- Payload: random, with parity of length based on the bit to send

ICMP Echo Scheme– Payload Length

Receive

```
[_Ipv4Hdr, IcmpHdr, Payload] =
  pkt:decapsulate(ipv4, Ip),
SeqNum = IcmpHdr#icmp.sequence,
IsNewByte = (SeqNum rem 8) =:= 0,
Payload =
  erlang:binary_to_list(Payload),
Bit = length(Payload) rem 2.
```

Then use IsNewByte and Bit somehow

ICMP Echo Scheme– Payload Length

We need to keep track of what state we're in:

- Waiting for a new byte (i.e., the low-order bit)
- Waiting for a higher-order bit

Also, the bits receive so far for the current byte

We can use a finite state machine – gen_fsm

Finite State Machines

- A system is in one of N states
- An event transitions the FSM to some other (possibly the same) state

Finite State Machines - Example

Digital keypad lock

- *State*: locked. *Event*: correct code entered.
  Transition: unlocked
- *State*: locked. *Event*: incorrect code entered.
  Transition: locked
- *State*: unlocked. *Event*: lock button pressed.
  Transition: locked
- *State*: unlocked. Event: timeout.
  Transition: locked

ICMP Echo Scheme– Payload Length - FSM

State: await_low_bit

Event: low order bit Bit arrives (evenly divisible by 8)

Action:

- `CurrentByte = Bit`
- Increment BitNum

Transition: await_higher_bit

ICMP Echo Scheme– Payload Length - FSM

State: await_higher_bit

Event: higher order bit Bit arrives

Action:

- `CurrentByte = CurrentByte bor (Bit bsl BitNum)`
  - If not the 8th bit, increment BitNum
  - Else, reset CurrentByte and BitNum and return complete byte to caller

Transition:

  - If not the 8th bit: await_higher_bit
  - Else: await_low_bit

ICMP Echo Scheme– Payload Length - FSM

State: await_low_bit

Event: higher order bit Bit arrives

Action:

- Ignore it (thus waiting for the next low-order bit)

Transition: await_low_bit

ICMP Echo Scheme– Payload Length - FSM

State: await_higher_bit

Event: low order bit Bit arrives

Action:

- Forget the partial data in CurrentByte
  - Reset CurrentByte and BitNum
  - Handle as would have been done if state had been await_low_bit

Transition: await_higher_bit

ICMP Echo Scheme– Payload Length

Demo:

```
icmp_data_receive_server:start_receive(
  payload_len).
icmp_data_send:send(
  "127.0.0.1", payload_len, “Shh!”).
```

Questions?

References

Base repository for this presentation

https://github.com/derek121/erlang_factory_2015

Steganography

https://www.cs.bham.ac.uk/~mdr/teaching/modules03/security/students/SS5/Steganography.htm

erl_img

https://github.com/evanmiller/erl_img

ICMP/Ping

http://en.wikipedia.org/wiki/Ping_%28networking_utility%29

References (cont.)

Appropriating Network Protocols

http://firstmonday.org/ojs/index.php/fm/article/view/528/449

gen_icmp

https://github.com/msantos/gen_icmp

gen_icmp Blog Post

http://blog.listincomprehension.com/2010/12/icmp-ping-in-erlang-part-2.html