Design and Implementation of an object-oriented, secure TCP/IP Stack

Hannes Mehnert, Andreas Bogk
23c3
27. December 2006
Overview

- Common software vulnerabilities
- Dylan
- Architecture of IP-Stack
CVE sorted by bug class

“Software Security is Software Reliability”, Felix Lindner, CACM 49/6
Data Reference Failures – Workarounds and solutions

- Buffer overflows:
  - Workarounds:
    - Stack canaries
    - Write xor execute
    - Randomized address spaces
  - Solution: Bounds checking

- Integer overflows:
  - Solution: bignums, exception on overflow

- Premature memory release
  - Solution: Automatic memory management
Input/Output Errors

- SQL injections
- Cross-site scripting
- Blue boxing
- 0-byte poisoning
- Perl OPEN
Interface Failures

- Format String Exploits
  - problem: varargs are not type safe
  - solution: language with type safe varargs

- Race conditions
OS Interface Flaws

- Directory Traversal
- Illegal File Access
- Remote Code Execution
Conclusion

- Prevention strategies exist for most bug classes
- Data Reference Failures can be avoided by choice of suitable programming language
Dylan

- Object oriented
- Functional aspects (higher order functions)
- Automatic memory management
- Dynamic and strong typing
- Bounds checks
- Optional type inference
- Supports encapsulation
- Features like scripting language (rapid prototyping), but compiled (performance)
Architecture for secure networking

- Packetizer – to parse and assemble protocols
  - Inspired by scapy
- Flow-Graph library – to specify flow of packets
  - Inspired by click
- Layering-mechanism – to stack protocols
  - Inspired by conduit+
let subseq = subsequence(bytes, start: 3 * 8, length: 8 * 10)

subsequence(subseq, start: 3 * 8, length: 8)
Frames

- ipv4-address
- mac-address
- count
- string
- string
- 0
- integer
- integer
- payload
- data
Frames – Size property

Fixed size

- ipv4-address
- mac-address

Variable size

- count
- string
- string 0
- payload
- data

Payload

Integer

- integer
- integer
Frames – Translation property

Untranslated

- ipv4-address
- mac-address

Translated

- count
- string
- string
- 0
- integer

- payload
- data
Container Frames

Name: destination-address
type: mac-address
static-start: 0
static-length: 6 * 8

Name: source-address
type: mac-address
static-start: 6 * 8
static-length: 6 * 8

Name: type-code
type: 2byte-big-endian-unsigned-integer
static-start: 12 * 8
static-length: 2 * 8

Name: payload
type: select (type-code)
#x800 => <ipv4-frame>
#x806 => <arp-frame>
end,
static-start: 14 * 8;
IPv4

- fixup: ceiling/(size(options) + 20, 4)
- fixup: frame.header-length + size(frame.payload)

<table>
<thead>
<tr>
<th>Version</th>
<th>Header Length</th>
<th>Type of Service</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification</td>
<td>flags</td>
<td>Fragmentation Offset</td>
<td></td>
</tr>
<tr>
<td>ttl</td>
<td>protocol</td>
<td>checksum</td>
<td></td>
</tr>
<tr>
<td>source-address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>destination-address</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Options and padding</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>payload</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- type-function: select (frame.protocol)
  - 6 => <tcp-frame>
  - 17 => <udp-frame>
- start: frame.header-length * 4 * 8,
- end: frame.length * 8;

- default-value: 4
Frame inheritance, repeated fields

Options field in ipv4-frame

<table>
<thead>
<tr>
<th>router-alert</th>
<th>timestamp</th>
<th>End-of-options</th>
<th>padding</th>
</tr>
</thead>
</table>

Frame inheritance, repeated fields

<table>
<thead>
<tr>
<th>flag</th>
<th>class</th>
<th>number</th>
</tr>
</thead>
</table>

ip-option-header

<table>
<thead>
<tr>
<th>router-alert (20)</th>
<th>length</th>
<th>value</th>
</tr>
</thead>
</table>

end-of-ip-options (0)

router-alert (20) length value
define protocol ethernet-frame (header-frame)
  field destination-address :: <mac-address>;
  field source-address :: <mac-address>;
  field type-code :: <2byte-big-endian-unsigned-integer>;
 variably-typed-field payload,
  type-function: select (frame.type-code)
    #x800 => <ipv4-frame>;
    #x806 => <arp-frame>;
    otherwise <raw-frame>;
end;
end;
Parsing ethernet-payload

source-address | destination | Type code | payload

00, de, ad, be, ef, 00, 00, 00, 00, 12, 23, 34, 08, 00, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1a, 1b, ...

parse type code
static start, static size

source-address | destination | #x800 | <ipv4-frame>, start: 14
### Parsing IPv4 payload – type

<table>
<thead>
<tr>
<th>version</th>
<th>Header length</th>
<th>Type of service</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identification</td>
<td>Fragmentation flags</td>
<td>Fragmentation offset</td>
<td></td>
</tr>
<tr>
<td>ttl</td>
<td>protocol</td>
<td>checksum</td>
<td></td>
</tr>
</tbody>
</table>

- **source-address**
- **destination-address**
- **Options and padding**
- **payload**
## Parsing IPv4 payload – boundaries

<table>
<thead>
<tr>
<th>version</th>
<th>Header length</th>
<th>Type of service</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Identification</td>
<td></td>
</tr>
<tr>
<td>ttl</td>
<td></td>
<td>protocol = 1</td>
<td>checksum</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fragmentation flags</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fragmentation offset</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>source-address</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>destination-address</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Options and padding</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>payload</td>
<td></td>
</tr>
</tbody>
</table>
Parsing payload of IPv4

<table>
<thead>
<tr>
<th>version</th>
<th>HL = 5</th>
<th>Type of service</th>
<th>length = 100</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Identification</td>
<td>Fragmentation flags</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ttl = 1</td>
<td>protocol = 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>source-address</td>
<td>destination-address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Options and padding</td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{<icmp-frame>}, \text{ start: } 5 \times 4, \text{ end: } 100
\]
define protocol ipv4-frame (header-frame)
  field version :: <4bit-unsigned-integer> = 4;
  field header-length :: <4bit-unsigned-integer>,
      fixup: ceiling/(reduce(\+, 20, map(frame-size, frame.options)), 4);
  field type-of-service :: <unsigned-byte> = 0;
  field total-length :: <2byte-big-endian-unsigned-integer>,
      fixup: frame.header-length * 4 + frame-size(frame.payload);
  field identification :: <2byte-big-endian-unsigned-integer> = 23;
  field evil :: <1bit-unsigned-integer> = 0;
  field dont-fragment :: <1bit-unsigned-integer> = 0;
  field more-fragments :: <1bit-unsigned-integer> = 0;
  field fragment-offset :: <13bit-unsigned-integer> = 0;
  field time-to-live :: <unsigned-byte> = 64;
  field protocol :: <unsigned-byte>;
  field header-checksum :: <2byte-big-endian-unsigned-integer> = 0;
  field source-address :: <ipv4-address>;
  field destination-address :: <ipv4-address>;
  repeated field options :: <ip-option-frame> = make(<stretchy-vector>),
      reached-end?: method(value :: <ip-option-frame>)
          instance?(value, <end-of-option-ip-option>)
  end;
 variably-typed-field payload,
  start: frame.header-length * 4 * 8,
  end: frame.total-length * 8,
  type-function: payload-type(frame);
end;
Parse code example

```haskell
let frame = parse-frame(<ipv4-frame>,
   packet: some-data);
format-out("Source address %=\n",
   frame.source-address);
```
**Assembly IPv4**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>Head length</td>
</tr>
<tr>
<td>Type of Service</td>
<td>flags</td>
</tr>
<tr>
<td>length</td>
<td>Fragmentation offset</td>
</tr>
<tr>
<td>Identification</td>
<td>protocol</td>
</tr>
<tr>
<td>ttl</td>
<td>checksum</td>
</tr>
<tr>
<td>source-address</td>
<td></td>
</tr>
<tr>
<td>destination-address</td>
<td></td>
</tr>
<tr>
<td>Options and padding</td>
<td></td>
</tr>
<tr>
<td>payload</td>
<td></td>
</tr>
</tbody>
</table>

User provided

Default value

fixup

fixup!
let v4-frame = make(<ipv4-frame>),
   source-address: ipv4-address("23.23.23.23"),
   destination-address: ipv4-address("42.42.42.42"),
   protocol: 23);
let byte-vector = assemble-frame(v4-frame).packet;
define method fixup! (frame :: <ipv4-frame>)
    frame.header-checksum := calculate-checksum(frame.packet);
    fixup!(frame.payload);
end;
Filter language

• Operators
  - And &
  - Or |
  - Not ~

• Rules
  - Presence of a frame-type (“ipv4”, “~ (dns)“)
  - Value of a field (“ipv4.destination-address = 23.23.23.23“)
  •“(udp.source-port = 53) | (udp.destination-port = 53)”
Flow-Graph

- **Filter**
  - One input
  - One output

- **Demultiplexer**
  - One input
  - Multiple outputs, each is associated with a filter rule

- **Fan-in**
  - Multiple inputs
  - One output

- **Closure-node**
  - One input
  - Executes closure with each packet received
let eth0 = make(<ethernet-interface>, name: "eth0");
connect(eth0, make(<summary-printer>));
toplevel(eth0);

Example: simple-sniffer

```
let eth0 = make(<ethernet-interface>, name: "eth0");
connect(eth0, make(<summary-printer>));
toplevel(eth0);
```
Example: simple-sniffer with filter

```
let eth0 = make(<ethernet-interface>, name: "eth0");
let filter = make(<frame-filter>, frame-filter: "arp");
connect(eth0, filter);
connect(filter, make(<summary-printer>));
toplevel(eth0);
```
Example: bridge

```
let eth0 = make(<ethernet-interface>, name: "eth0");
let eth1 = make(<ethernet-interface>, name: "eth1");
connect(eth0, eth1);
connect(eth1, eth0);
make(<thread>, function: curry(toplevel, eth0));
make(<thread>, function: curry(toplevel, eth1));
```
Layering – ethernet layer
ARP

arp-handler

ethernet-address = 00:de:ad:be:ef:00

arp-table

decapsulator

filter rule
“ethernet.type-code = #x806”

template frame

type-code = #x806
source-address = 00:de:ad:be:ef:00

completer

ethernet-address = 00:de:ad:be:ef:00

filter rule
“ethernet.type-code = #x806”

ethernet-layer

demultiplexer

fan-in

eth0
IP-over-ethernet Adapter

- ip-layer
- arp-handler
- ip-over-ethernet-adapter
- ip-socket #x800
- arp-socket #x806
- ethernet-layer
Applications

- Protocols currently supported
  - Ethernet, IPv4, TCP, UDP, DNS, 802.11, ARP, pcap, STP, DHCP, ID3v2
- Secure Network Services
- Firewall
- Network Night Vision
- Intrusion detection Systems
Conclusion

- Software Architecture is Software Security
- IP Stack without remote exploits
Announcements

- http://www.networknightvision.com
- Dylan Hackers Conference Europe 2006
- 15:30 – 17:00 Dylan Introduction Workshop
- 21:30 – 23:30 packetizer workshop
Links

- Dylan Website [http://www.opendylan.org](http://www.opendylan.org)
- Software Security is Software Reliability [http://doi.acm.org/10.1145/1132469.1132502](http://doi.acm.org/10.1145/1132469.1132502)
- Click [http://www.read.cs.ucla.edu/click/](http://www.read.cs.ucla.edu/click/)
  - Hueni, Johnson, Engel, OOPSLA '95
- Code: svn://svn.opendylan.org/scm/svn/dylan/trunk/libraries
  - (especially packetizer, sniffer, flow, layer)