Security Testing using Software Transformation

Thomas Dean
Queen's University

Scott Knight
Royal Military College of Canada
Collaborative work between Scott Knight and myself at the Royal Military College of Canada

Initially based on a project to test an X.509 (Public Key) implementation that was to be deployed on the classified network within DND (Canada).

Similar to the PROTOS project at the Oulu University.
Basic Premises

- Software Developers are always under a great deal of time and budget constraints when developing software (Time to Market).

- There are often implementation constraints
  - memory footprint
  - limited CPU performance
  - demanding throughput requirements

- Implementations that meet conformance tests may have open vulnerabilities to more obscure cases that might not even be possible during the normal operation of the protocol.
Basic Premises

- Protocols are complex systems.
  - it is a *mistake* to treat them only as data
  - Protocols have their own syntax and semantics
  - Semantics include constraints that must be implemented correctly

- Apply language analysis techniques to the protocol

- Generating test sets can be implemented using transformation tools and techniques

- Goal: lightweight approaches that leverage source analysis tools
Obtain a valid PDU from the network
Obtain a valid PDU from the network
Generate multiple mutant PDUs
Obtain a valid PDU from the network
Generate multiple mutant PDUs
Inject mutant PDUs back into the network
Obtain a valid PDU from the network
Generate multiple mutant PDUs
Inject mutant PDUs back into the network
Between each mutant PDU, verify the health of the target with the valid PDU
Mutation

- Protocols have two levels
  - data structure (syntax/semantic) level
  - transfer encoding level
- Changes can be made at both levels
- Markup/Execution Separation
  - Start by annotating the textual representation of the PDU with markups indicating the changes/alternate encodings that should be made
  - Separate phase to actually change the data to implement the changes
Example ASN.1 Spec

HDP_PDU ::= SEQUENCE {
    number_of_houses INTEGER
    houses Houses
}

Houses ::= SEQUENCE OF House

House: ::= SEQUENCE {
    house_number INTEGER
    family_name VisibleString
}
HDP_PDU SEQUENCE {
    number_of_houses INT 2
    houses SEQUENCE {
        houses*1 SEQUENCE {
            house_number INT 200
            family_name VisibleString “Smith”
        }
        houses*2 SEQUENCE {
            house_number INT 300
            family_name VisibleString “Stevens”
        }
    }
}
HDP_PDU SEQUENCE {
  number_of_houses INT 2
  houses SEQUENCE {
    houses*1 SEQUENCE {
      house_number INT 200
      family_name VisibleString "Smith"
    }
    houses*2 SEQUENCE {
      house_number INT 300
      family_name VisibleString "Stevens"
    }
  }
}
HDP_PDU SEQUENCE {
  number_of_houses INT 2
  houses SEQUENCE {
    houses*1 SEQUENCE {
      house_number INT ErrASN remove 200
      family_name VisibleString “Smith”
    }
    houses*2 SEQUENCE {
      house_number INT 300
      family_name VisibleString “Stevens”
    }
  }
}

HDP_PDU SEQUENCE  ErrDER increase_length 400 {
  number_of_houses INT 2
  houses SEQUENCE {
    houses*1 SEQUENCE {
      house_number INT ErrASN remove 200
      family_name  VisibleString “Smith”
    }
    houses*2 SEQUENCE {
      house_number INT 300
      family_name  VisibleString “Stevens”
    }
  }
}
Constraint Error

HDP_PDU SEQUENCE  ErrDER increase_length 400 {
  number_of_houses INT  ErrASN NewValue 3 2
  houses SEQUENCE {
    houses*1 SEQUENCE {
      house_number INT  ErrASN remove 200
      family_name   VisibleString “Smith”
    }
    houses*2 SEQUENCE {
      house_number INT 300
      family_name   VisibleString “Stevens”
    }
  }
}
HDP_PDU SEQUENCE  ErrDER increase_length 400 
   number_of_houses INT 3 
   houses SEQUENCE 
      houses*1 SEQUENCE 
         family_name VisibleString “Smith” 
      } 
   houses*2 SEQUENCE 
      house_number INT 300 
      family_name VisibleString “Stevens” 
   } 
}
Types of Errors

- Syntax Errors
  - removing fields
  - swapping fields
  - duplicating fields
- Value Errors
  - Value mismatch (e.g. type fields)
  - Range errors
  - Format errors (date, IP address)
- Encoding Errors
  - Length errors
  - Unusual lengths
  - DER errors
Experience

- TFTP, SNMP
- x.509 Public Key Library
- OSPF
HDP_PDU ::= SEQUENCE {
    number_of_houses INTEGER
    houses                 Houses
}

Houses ::= SEQUENCE OF House

House: := SEQUENCE {
    house_number    INTEGER
    family_name     VisibleString
}

This field give the length of

Currently specified in textual prose in the ASN.1 Spec
HDP_PDU ::= SEQUENCE {
    number_of_houses INTEGER
    houses Houses
}
<TRANSFER>number_of_houses is 2 bytes</TRANSFER>
<CONSTRAINT>
    Cardinality(houses) = number_of_houses
</CONSTRAINT>
Houses ::= SEQUENCE OF House
House ::= SEQUENCE {
    house_number INTEGER
    family_name VisibleString
}
<TRANSFER>house_number is 2 bytes
    family_name is 7 bytes</TRANSFER>
HDP_PDU ::= SEQUENCE {
    number_of_houses        INTEGER
    houses                  Houses
}

<TRANSFER>number_of_houses is 2 bytes</TRANSFER>

<CONSTRAINT>
    Cardinality(houses) = number_of_houses
</CONSTRAINT>

Houses ::= SEQUENCE OF House

House ::= SEQUENCE {
    house_number  INTEGER
    family_name   VisibleString
}

<TRANSFER>house_number is 2 bytes
    family_name is 7 bytes</TRANSFER>
HDP_PDU ::= SEQUENCE {
    number_of_houses INTEGER
    houses         Houses
}
<TRANSFER>number_of_houses is 2 bytes</TRANSFER>
<CONSTRAINT>
    Cardinality(houses) = number_of_houses
</CONSTRAINT>
Houses ::= SEQUENCE OF House
House: ::= SEQUENCE {
    house_number   INTEGER
    family_name    VisibleString
}
<TRANSFER>house_number is 2 bytes
    family_name is 7 bytes</TRANSFER>
SCL Constraints

- Value
- Range
- Enumerated
- Cardinality
- Length
- Order (Sorted)
- Unique
**Mutator Framework**

Diagram showing the process:

- **Good PDU**
  - **Decoder**
  - **Textual PDU**
    - **Protocol Description**
      - **Extractor**
        - **Protocol Model**
          - **Test Planning**
            - **Test Database**

- **Mutant Markup PDUs**
  - **Markup**
  - **Execution**
    - **Mutant PDUs (Text)**
      - **Encoder**
      - **Mutant PDUs (binary)**
Mutator Framework

- Good PDU
- Decoder
- Textual PDU
- Protocol Description
- Extractor
- Protocol Model
- Test Planning
- Test Database
- Script
- Mutant Markup PDUs
- Execution
- Mutant PDUs (Text)
- Encoder
- Mutant PDUs (binary)
Mutator Framework

- Good PDU
- Decoder
- Textual PDU
- Protocol Description
- Extractor
- Protocol Model
- Test Planning
- Test Database
- Script
- Mutant Markup PDUs
- Execution
- Mutant PDUs
  (Text)
- Encoder
- Mutant PDUs
  (binary)
Mutator Framework

Good PDU → Decoder → Textual PDU → Protocol Description → Extractor

Protocol Description → Extractor

Extractor → Protocol Model

Protocol Model → Test Planning

Test Planning → Test Database

Test Database

Mutant Markup PDUs → Script

Script

Test Planning

Execution

Execution

Mutant Markup PDUs (Text)

Mutant Markup PDUs (Text)

Mutant PDUs (Text)

Mutant PDUs (Text)

Mutant PDUs (binary)

Encoder

Encoder
State Based Protocols

- Current Version is Request-Response
  - OSPF, BGP, TFTP, SNMP, x509 Certs
  - Value of One exchange does not depend on previous exchanges
- Constraints between PDUs
  - Persistent Elements
  - Conversation
State Explosion

Pkt1 → Rsp1 → Pkt2b → Rsp2b → Pkt3bj → Rsp3bj → ...

Pkt2a → Rsp2a → ...

Pkt2c → Rsp2c → ...

Pkt3bi → Rsp3bi → ...

Pkt3bk → Rsp3bk → ...
Higher Order Attribute Grammars
- User writes grammar that contains the planned sequence to test
  - both send and receive PDUs
- Java based actions provide state based transfers
- Scripts generate variants for testing
- Walker traverses grammar tree, triggering actions
- Executable specification of a client or server
Formal State Models

- Used in many Intrusion Detection systems
- Some protocols are specified using state models
  - can we leverage this?
- State model describes valid sequences of PDUs and the constraints between them
- Still building an executable model, still a lot of work!!
Use the Existing Software

- We already have an executable specification of the protocol
  - client and server both pass conformance tests, we are testing security
- Don’t need to specify entire tree of protocol exchanges, client and server can do it for you!!
Capture Sequence

Pkt1  Rsp1  Pkt2  Rsp2  Pkt3  Rsp3
Mutate One of Sequence
The Test

- Replay the sequence of packets
  - But must be able to handle state data such as file handle for an open file in file sharing protocol
- Don’t want to have to implement a testing version of client or server!!
  - Don’t want to have to decode packets or each time through the test
- Based on repeatability (PDUs will be the almost identical each time)
Capture Sequence

Pkt1 → Rsp1 → Pkt2 → Rsp2 → Pkt3 → Rsp3
**Injector Script**

- Copy values from one packet to another
  - Possibly with some function
- Specified by byte offset within packet
  - Injector is very simple

password == "zhang";

sendPK.2.47.51 == recPK.1.52.56;

sendPK.2.65.88 == SMBencrypt(password, recPK.1.73.80);

sendPK.6.37.38 == recPK.5.37.38
Protocol Analysis

- How do we get the injector script?
  - Augment SCL with dependencies between packets
  - ReadCmd.FileNum == OpenResp.FileNum
- Decode and parse captured packets
- Use dependencies from protocol description and filter by actual packets capture
  - derive byte offsets from actual captured fields
Protocol Analysis

Advantage:
- State dependencies become simple data dependencies
- Simple copy and computation functions

Disadvantages
- Errors requiring sequence of two mutated packets cannot be found
- Some difficulties with variant length fields
Open Problems

- Text Based Protocols
- Currently Binary Protocols
- SMTP, HTTP, SOAP
- More Traditional Grammars
- Agile Parsing to Introduce Errors
  (Aspect Grammars??)
- Multiple Layers (Browser → Web Server → CGI/PHP/J2EE → Database)
Other Issues

- Code Analysis
- Test–to–Code
  - Many current techniques
  - Test data generated from a semantic constraint. Does this help us?
- Code–to–Test
  - Picking Values for Changes
  - Lightweight Analysis
Questions?