Automated Attack Synthesis by Extracting Finite State Machines from Protocol Specification Documents

Maria L. Pacheco*, Max von Hippel®, Ben Weintraub®, Dan Goldwasser*, Cristina Nita-Rotaru®

{pachecog, dgoldwas}@purdue.edu
{vonhippel.m, weintraub.b, c.nitarotaru}@northeastern.edu

* Purdue University, © Northeastern University. Image courtesy of Wikimedia. This work was supported by NSF grants CNS-1814105, CNS-1815219, and GRFP-1938052.
Automated Protocol Analysis

The internet runs on protocols, like TCP, UDP, DCCP, SFTP, etc.
Automated Protocol Analysis

The internet runs on protocols, like TCP, UDP, DCCP, SFTP, etc.

Each protocol peer runs a finite state machine.
Automated Protocol Analysis

The internet runs on protocols, like TCP, UDP, DCCP, SFTP, etc.

Each protocol peer runs a finite state machine.

Protocol flaws are found by analyzing the FSM.

Image courtesy of Scientific American.
From Spec to Implementation

RFC Specification

The nine possible states are as follows. They are listed in increasing order, so that "state => CLOSSEQ" means the same as "state = CLOSSEQ or state = CLOSING or state = TIMEDWAIT". Section 8 describes the states in more detail.

CLOSED
Represents nonexistent connections.

LISTEN
Represents server sockets in the passive listening state. LISTEN and CLOSED are not associated with any particular DCCP connection.

REQUEST
A client socket enters this state, from CLOSED, after sending a DCCP-Request packet to try to initiate a connection.

RESPOND
A server socket enters this state, from LISTEN, after receiving a DCCP-Request from a client.

PARKHOP
A client socket enters this state, from REQUEST, after receiving a DCCP-Response from the server. This state represents the third phase of the three-way handshake. The client may send application data in this state, but it MUST include an Acknowledgement Number on all of its packets.

● Produced by IETF.
● Written in English prose.
The nine possible states are as follows. They are listed in increasing order, so that "state == CLOSEDREQ" means the same as "state = CLOSING or state = TIMELAST". Section 8 describes the states in more detail.

CLOSED
Represents nonexistent connections.

LISTEN
Represents server sockets in the passive listening state. LISTEN and CLOSED are not associated with any particular DCCP connection.

REQUEST
A client socket enters this state, from CLOSED, after sending a DCCP-Request packet to try to initiate a connection.

RESPONSE
A server socket enters this state, from LISTEN, after receiving a DCCP-Request from a client.

PASSTHR
A client socket enters this state, from REQUEST, after receiving a DCCP-Response from the server. This state represents the third phase of the three-way handshake. The client may send application data in this state, but it MUST include an Acknowledgement Number on all of its packets.

From Spec to Implementation

- Produced by IETF.
- Written in English prose.

Implementation

- Written in C, Go, Rust, etc. by a programmer.
From Spec to Implementation

RFC Specification

The nine possible states are as follows. They are listed in increasing order, so that "state => CLOSEREQ" means the same as "state = CLOSEREQ or state = CLOSING or state = TIMWAIT". Section 8 describes the states in more detail.

CLOSED
Represents nonexistent connections.

LISTEN
Represents server sockets in the passive listening state. LISTEN and CLOSED are not associated with any particular DCCP connection.

REQUEST
A client socket enters this state, from CLOSED, after sending a DCCP-Request packet to try to initiate a connection.

RESPOND
A server socket enters this state, from LISTEN, after receiving a DCCP-Request from a client.

PARTOPEN
A client socket enters this state, from REQUEST, after receiving a DCCP-Response from the server. This state represents the third phase of the three-way handshake. The client may send application data in this state, but it MUST include an Acknowledgement Number on all of its packets.

Implementation

● How does the programmer interpret the specification?
Where do Bugs Come From?

RFC Specification

The nine possible states are as follows. They are listed in increasing order, so that "state == CLOSEREQ" means the same as "state = CLOSING or state = TIMENOUT". Section 8 describes the states in more detail.

CLOSED
Represents nonexistent connections.

LISTEN
Represents server sockets in the passive listening state. LISTEN and CLOSED are not associated with any particular DCCP connection.

REQUEST
A client socket enters this state, from CLOSED, after sending a DCCP-Request packet to try to initiate a connection.

RESPOND
A server socket enters this state, from LISTEN, after receiving a DCCP-Request from a client.

FINISHED
A client socket enters this state, from REQUEST, after receiving a DCCP-Response from the server. This state represents the third phase of the three-way handshake. The client may send application data in this state, but it MUST include an Acknowledgement Number on all of its packets.

Fundamental issues with the protocol design.
Where do Bugs Come From?

Fundamental issues with the protocol design.

Ambiguities and omissions in the specification.

RFC Specification

The nine possible states are as follows. They are listed in increasing order, so that "state => CLOSEREQ" means the same as "state = CLOSEREQ or state = CLOSING or state = TIMEMOUT". Section 8 describes the states in more detail.

**CLOSED**
Represent nonexistent connections.

**LISTEN**
Represents server sockets in the passive listening state. LISTEN and CLOSED are not associated with any particular DCCP connection.

**REQUEST**
A client socket enters this state, from CLOSED, after sending a DCCP-Request packet to try to initiate a connection.

**RESPOND**
A server socket enters this state, from LISTEN, after receiving a DCCP-Request from a client.

**PARTOPEN**
A client socket enters this state, from REQUEST, after receiving a DCCP-Response from the server. This state represents the third phase of the three-way handshake. The client may send application data in this state, but it MUST include an Acknowledgment Number on all of its packets.
Where do Bugs Come From?

RFC Specification

- The nine possible states are as follows. They are listed in increasing order, so that "state -> CLOSEREQ" means the same as "state = CLOSEREQ or state = CLOSING or state = TIMWAIT". Section 8 describes the states in more detail.
  - CLOSED
    - Represents nonexistent connections.
  - LISTEN
    - Represents server sockets in the passive listening state. LISTEN and CLOSED are not associated with any particular DCCP connection.
  - REQUEST
    - A client socket enters this state, from CLOSED, after sending a DCCP-Request packet to try to initiate a connection.
  - RESPOND
    - A server socket enters this state, from LISTEN, after receiving a DCCP-Request from a client.
  - FARTOPEN
    - A client socket enters this state, from REQUEST, after receiving a DCCP-Response from the server. This state represents the third phase of the three-way handshake. The client may send application data in this state, but it MUST include an Acknowledgment Number on all of its packets.

Implementation

- Programming mistakes.
- Ambiguities and omissions in the specification.
- Fundamental issues with the protocol design.
Where do Bugs Come From?

RFC Specification

The nine possible states are as follows. They are listed in increasing order, so that "state => CLOSEREQ" means the same as "state = CLOSEREQ or state = CLOSING or state = TIMEWAIT". Section 8 describes the states in more detail.

CLOSED
Represents nonexistent connections.

LISTEN
Represents server sockets in the passive listening state. LISTEN and CLOSED are not associated with any particular DCCP connection.

REQUEST
A client socket enters this state, from CLOSED, after sending a DCCP-Request packet to try to initiate a connection.

RESPOND
A server socket enters this state, from LISTEN, after receiving a DCCP-Request from a client.

PARTOPEN
A client socket enters this state, from REQUEST, after receiving a DCCP-Response from the server. This state represents the third phase of the three-way handshake. The client may send application data in this state, but it MUST include an Acknowledgment Number on all of its packets.

Implementation

Programming mistakes.
Where do Bugs Come From?

RFC Specification

The nine possible states are as follows. They are listed in increasing order, so that "state => CLOSEREQ" means the same as "state = CLOSEREQ or state = CLOSING or state = TIMEWAIT". Section 8 describes the states in more detail.

CLOSED
Represents nonexistent connections.

LISTEN
Represents server sockets in the passive listening state. LISTEN and CLOSED are not associated with any particular DCCP connection.

REQUEST
A client socket enters this state, from CLOSED, after sending a DCCP-Request packet to try to initiate a connection.

RESPOND
A server socket enters this state, from LISTEN, after receiving a DCCP-Request from a client.

PAROPEN
A client socket enters this state, from REQUEST, after receiving a DCCP-Response from the server. This state represents the third phase of the three-way handshake. The client may send application data in this state, but it MUST include an Acknowledgement Number on all of its packets.

Implementation

Property Testers
Heuristic Algorithms
Protocol Bake-Offs

Symbolic or Concolic Execution
Randomized Testing
Static or Dynamic Analysis

Programming mistakes.

Fuzzing

... etc.
RFC Specification

The nine possible states are as follows. They are listed in increasing order, so that "state => CLOSEREQ" means the same as "state = CLOSEREQ or state = CLOSING or state = TIMELAPSE". Section 8 describes the states in more detail.

CLOSED
- Represents nonexistent connections.

LISTEN
- Represents server sockets in the passive listening state. LISTEN and CLOSED are not associated with any particular DCCP connection.

REQUEST
- A client socket enters this state, from CLOSED, after sending a DCCP-Request packet to try to initiate a connection.

RESPOND
- A server socket enters this state, from LISTEN, after receiving a DCCP-Request from a client.

FASTOPEN
- A client socket enters this state, from REQUEST, after receiving a DCCP-Response from the server. This state represents the third phase of the three-way handshake. The client may send application data in this state, but it MUST include an Acknowledgment Number on all of its packets.

Fundamental issues with the protocol design.

Implementation

Ambiguities and omissions in the specification.
This Presentation

RFC Specification

The nine possible states are as follows. They are listed in increasing order, so that "state => CLOSEDSEQ" means the same as "state = CLOSEDSEQ or state = CLOSING or state = TIMEWAIT". Section 8 describes the states in more detail.

CLOSED
- Represents nonexistent connections.

LISTEN
- Represents server sockets in the passive listening state. LISTEN and CLOSED are not associated with any particular DCCP connection.

REQUEST
- A client socket enters this state, from CLOSED, after sending a DCCP-Request packet to try to initiate a connection.

RESPOND
- A server socket enters this state, from LISTEN, after receiving a DCCP-Request from a client.

PARtimeout
- A client socket enters this state, from REQUEST, after receiving a DCCP-Response from the server. This state represents the third phase of the three-way handshake. The client may send application data in this state, but it MUST include an Acknowledgement Number on all of its packets.

FSM Interpretation

NLP

NLP
This Presentation

RFC Specification

The nine possible states are as follows. They are listed in increasing order, so that "state => CLOSEDSEQ" means the same as "state => CLOSING or state => TIMEWAIT". Section 8 describes the states in more detail.

CLOSED
- Represents nonexistent connections.

LISTEN
- Represents server sockets in the passive listening state. LISTEN and CLOSED are not associated with any particular DCCP connection.

REQUEST
- A client socket enters this state, from CLOSED, after sending a DCCP-Request packet to try to initiate a connection.

RESPOND
- A server socket enters this state, from LISTEN, after receiving a DCCP-Request from a client.

PASTOPEN
- A client socket enters this state, from REQUEST, after receiving a DCCP-Response from the server. This state represents the third phase of the three-way handshake. The client may send application data in this state, but it MUST include an Acknowledgement Number on all of its packets.

FSM Interpretation

Bugs & Attacks

NLP

Attack Synthesis
Extracting FSMs from RFCs

RFC Specification

The nine possible states are as follows. They are listed in increasing order, so that "state \( \Rightarrow \) CLOSED\textsc{REQ}" means the same as "state \( \Rightarrow \) CLOS\textsc{ING}" or state \( \Rightarrow \) TIME\textsc{WAIT}". Section 8.2 describes the states in more detail.

CLOSED
- Represents nonexistent connections.

LISTEN
- Represents server sockets in the passive listening state. LISTEN and CLOSED are not associated with any particular DCCP connection.

REQUEST
- A client socket enters this state, from CLOSED, after sending a DCCP-Request packet to try to initiate a connection.

RESPOND
- A server socket enters this state, from LISTEN, after receiving a DCCP-Request from a client.

PARSTOP
- A client socket enters this state, from REQUEST, after receiving a DCCP-Response from the server. This state represents the third phase of the three-way handshake. The client may send application data in this state, but it MUST include an Acknowledgement Number on all of its packets.

FSM Interpretation

Bugs & Attacks

NLP

Attack Synthesis
Extracting FSMs from RFCs: Main Challenges

- **No one-to-one mapping** between the text and the canonical FSM

The nine possible states are as follows. They are listed in increasing order, so that "state \( \Rightarrow \) CLOSEREQ" means the same as "state \( \Rightarrow \) CLOSING or state \( \Rightarrow \) TIMWAIT". Section II describes the states in more detail.

- **CLOSED**: Represents nonexistent connections.
- **LISTEN**: Represents server sockets in the passive listening state. LISTEN and CLOSED are not associated with any particular DCCP connection.
- **REQUEST**: A client socket enters this state, from CLOSED, after sending a DCCP-Request packet to try to initiate a connection.
- **RESPOND**: A server socket enters this state, from LISTEN, after receiving a DCCP-Request from a client.
- **PARTOPEN**: A client socket enters this state, from REQUEST, after receiving a DCCP-Response from the server. This state represents the third phase of the three-way handshake. The client may send application data in this state, but it MUST include an Acknowledgement Number on all of its packets.
Extracting FSMs from RFCs: Main Challenges

- No one-to-one mapping between the text and the canonical FSM
- RFCs contain **omissions, mistakes, & ambiguities.**

The client leaves the PARTOPEN state for OPEN when it receives a valid packet other than DCCP-Response, DCCP-Reset, or DCCP-Sync from the server.

Why not [PARTOPEN – DCCP-Close? → OPEN]
Extracting FSMs from RFCs: Main Challenges

- There is no canonical FSM.
- RFCs contain omissions, mistakes, & ambiguities.
- Off-the-shelf NLP approaches are not suitable.
Extracting FSMs from RFCs: Main Challenges

- There is no canonical FSM.
- RFCs contain omissions, mistakes, & ambiguities.
- Off-the-shelf NLP approaches are not suitable.
- There is a lot of variation in the language and structure of different RFCs.
Our Approach

(1) Technical Language Embedding
Our Approach

(1) Technical Language Embedding

(2) Zero-shot Protocol Information Extraction
Our Approach

(1) Technical Language Embedding

(2) Zero-shot Protocol Information Extraction

(3) Protocol State Machine Extraction
Our Approach

(1) Technical Language Embedding
(2) Zero-shot Protocol Information Extraction
(3) Protocol State Machine Extraction
(4) Applications

Monitoring
Fuzzing
Attack Synthesis
Program Analysis
Model Checking
Step 1. Learning Technical Language Embeddings

- Contextualized representations

The connection is in error and should be reset vs. Send Reset Code 5

Each word is informed by all of its surroundings

Trained on 8,858 documents and approximately 475M words
Step 2. Protocol Information Extraction

REQUEST
A client socket enters this state, from CLOSED, after sending a DCCP-Request packet to try to initiate a connection.
Step 2. Zero-Shot Protocol Information Extraction: **Grammar**

- **Definition tags**, used to define states, events, etc.;
Step 2. Zero-Shot Protocol Information Extraction: **Grammar**

- **Definition tags**, used to define states, events, etc.;
- **Reference tags**, used to observe mentions of previously defined data;
Step 2. Zero-Shot Protocol Information Extraction: **Grammar**

- **Definition tags**, used to define states, events, etc.;
- **Reference tags**, used to observe mentions of previously defined data;
- **State Machine tags**, used to track transitions, actions, etc;
Step 2. Zero-Shot Protocol Information Extraction: Grammar

- **Definition tags**, used to define states, events, etc.;
- **Reference tags**, used to observe mentions of previously defined data;
- **State Machine tags**, used to track transitions, actions, etc;
- **Control flow tags**, used to record the logical structure of the FSM.

```plaintext
bool ::= true | false
type ::= send | receive | issue
def-tag ::= def_state | def_var | def_event
ref-state ::= ref_state id="##"
ref-event ::= ref_event id="##" type="type"
ref-tag ::= ref-event | ref-state
def-atom ::= <def-tag>engl</def-tag>
sm-atom ::= <ref-tag>engl</ref-tag> | engl
sm-tag ::= trigger | variable | error | timer
act-atom ::= <arg>sm-atom</arg> | sm-atom
act-struct ::= act-struct | act-struct act-atom
trn-arg ::= arg_source | arg_target | arg_inter
trn-atom ::= <trn-arg>sm-atom<trn-arg> | sm-atom
trn-struct ::= trn-struct | trn-struct trn-atom
ctl-atom ::= <sm-tag>sm-atom</sm-tag>
  | <action type="type">act-struct</action>
  | <transition>trn-struct</transition>
  | sm-atom
ctl-struct ::= ctl-atom | ctl-struct ctl-atom
ctl-rel ::= relevant=bool
control ::= <control ctl-rel>ctl-struct</control>
e ::= control | ctl-atom | def-atom | e_0 e_1
```

<control relevant="true">
  <transition>
    The client leaves the
    <arg_source>
      <ref_state id="3">REQUEST</ref_state>
    </arg_source>
    state for
    <arg_target>
      <ref_state id="5">PARTOPEN</ref_state>
    </arg_target>
  </transition>
  <trigger>
    when it receives a
    <ref_event type="receive" id="2">
      DCCP-Response
    </ref_event>
    from the server.
  </trigger>
</control>

Control block scopes search.
The client leaves the REQUEST state for PARTOPEN when it receives a DCCP-Response from the server.
The client leaves the REQUEST state for PARTOPEN when it receives a DCCP-Response from the server.

The client leaves the REQUEST state for PARTOPEN when it receives a DCCP-Response from the server.

The client leaves the `<ref_state id="3">REQUEST</ref_state>` state for `<ref_state id="5">PARTOPEN</ref_state>` when it receives a `<ref_event type="receive" id="2">DCCP-Response</ref_event>` from the server.
The client leaves the \texttt{REQUEST} state for the \texttt{PARTOPEN} state when it receives a DCCP-Response from the server.

```
<control relevant="true">
  <transition>
    The client leaves the
    <arg_source>
      <ref_state id="3">REQUEST</ref_state>
    </arg_source>
    state for
    <arg_target>
      <ref_state id="5">PARTOPEN</ref_state>
    </arg_target>
  </transition>

  <trigger>
    when it receives a
    <ref_event type="receive" id="2">DCCP-Response</ref_event>
    from the server.
  </trigger>
</control>
```

Control block scopes search.

Transition block contains a transition $s \rightarrow s'$.

Source state $s$ is described in arg source within a state reference.

Target state $s'$ is described in arg target within a state reference.

The transition is triggered by an event.

In the event, the peer receives the packet DCCP-Response, which we assign identifier “2”.

37
Step 2. Zero-Shot Protocol Information Extraction: LinearCRF

1. Split text in chunks

If SND.UNA > ISS
change the connection state to ESTABLISHED.
Step 2. Zero-Shot Protocol Information Extraction: LinearCRF

2. Extract features

If SND.UNA > ISS
change the connection state to ESTABLISHED.
Step 2. Zero-Shot Protocol Information Extraction: **LinearCRF**

### Linear CRF Model

$$p(y|x) = \frac{p(y,x)}{\sum_{y'} p(y',x)}$$

$$p(x, y) = \prod_{t=1}^{T} \exp(f(y_t, y_{t-1}, x_t; \theta))$$

- **B-Trigger**
  - If SND.UNA > ISS

- **B-Transition**
  - change the connection state

- **I-Transition**
  - to ESTABLISHED.
Step 2. Zero-Shot Protocol Information Extraction: **NeuralCRF**

![Diagram](image)

**3. BiLSTM-CRF**

\[
p(y|x) = \frac{p(y, x)}{\sum_{y'} p(y', x)}
\]

\[
p(x, y) = \prod_{t=1}^{T} \exp(f(y_t, y_{t-1}, x_t; \theta))
\]

\[
f(y_t, y_{t-1}, x_t) = h_t + P_{y_t, y_{t-1}}
\]

If SND.UNA > ISS, change the connection state to ESTABLISHED.
Step 2. Zero-Shot Protocol Information Extraction: **Evaluation**

Heuristics based on word usage:

- `leave/move` → `transition`
- `send/receive/issue` → `action`
- `if/while` → `trigger`

...
Step 2. Zero-Shot Protocol Information Extraction: **Evaluation**

The neural model **outperforms** the linear model.
Step 3. Protocol State Machine Extraction

<control relevant="true">
  <transition>
    The client leaves the
    <arg_source>
      <ref_state id="3">REQUEST</ref_state>
    </arg_source>
    state for
    <arg_target>
      <ref_state id="5">PARTOPEN</ref_state>
    </arg_target>
  </transition>

  <trigger>
    when it receives a
    <ref_event type="receive" id="2">DCCP-Response</ref_event>
    from the server.
  </trigger>
</control>

Control block scopes search.

Transition block contains a transition \( s \rightarrow s' \).

Source state \( s \) is described in arg source within a state reference.

Target state \( s' \) is described in arg target within a state reference.

The transition is triggered by an event.

In the event, the peer receives the packet DCCP-Response, which we assign identifier “2”.
Step 3. Protocol State Machine Extraction

Control block scopes search.

Transition block contains a transition $s \rightarrow s'$.

Source state $s$ is described in arg source within a state reference.

Target state $s'$ is described in arg target within a state reference.

The transition is triggered by an event.

In the event, the peer receives the packet DCCP-Response, which we assign identifier “2”.

The client leaves the REQUEST state for PARTOPEN when it receives a DCCP-Response from the server.
Step 3. Protocol State Machine Extraction

Heuristic Extraction Alg.

- Search lower for target states
- Search higher for source states
- Search higher (< 7 layers) for event(s)
- Handle set complement
- Heuristically prune bad transitions
Step 3. Protocol State Machine Extraction - TCP

TCP FSM: Extracted Transitions

- Gold
- LinearCRF+R
- NeuralCRF+R

- ~Correct
- Incorrect
Step 3. Protocol State Machine Extraction - **TCP**

Transitions that are **good enough** for our FSM:
- Correct **source state**
- Correct **target state**
- At least **one correct event**
Step 3. Protocol State Machine Extraction - **TCP**

TCP FSM: Extracted Transitions

<table>
<thead>
<tr>
<th>Method</th>
<th>~Correct</th>
<th>Incorrect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gold</td>
<td>16</td>
<td>0</td>
</tr>
<tr>
<td>LinearCRF+R</td>
<td>15</td>
<td>5</td>
</tr>
<tr>
<td>NeuralCRF+R</td>
<td>15</td>
<td>5</td>
</tr>
</tbody>
</table>

Represents our **skyline**: The best we can do with our **gold intermediary rep.**
Step 3. Protocol State Machine Extraction - TCP

We recover most transitions

Linear and Neural FSMs are identical.
Step 3. Protocol State Machine Extraction - **DCCP**
Step 3. Protocol State Machine Extraction - **Missing Transition**

Consider the transition:

CLOSE_WAIT --- FIN! ---＞ LAST_ACK

Described in the RFC as follows:

**CLOSE-WAIT STATE**

Since the remote side has already sent FIN, RECEIVES must be satisfied by text already on hand, but not yet delivered to the user. If no text is awaiting delivery, the RECEIVE will get a "error: connection closing" response. Otherwise, any remaining text can be used to satisfy the RECEIVE.
Step 3. Protocol State Machine Extraction - **Missing Transition**

Consider the transition:

CLOSE_WAIT --- **FIN!** ----> LAST_ACK

Described in the RFC as follows:

```
CLOSE-WAIT STATE

Since the remote side has already sent FIN, RECEIVES must be satisfied by text already on hand, but not yet delivered to the user. If no text is awaiting delivery, the RECEIVE will get a "error: connection closing" response. Otherwise, any remaining text can be used to satisfy the RECEIVE.
```
Step 3. Protocol State Machine Extraction - **Missing Transition**

Consider the transition:

```markdown
CLOSE_WAIT --- FIN! --- LAST_ACK
```

Described in the RFC as follows:

Since the remote side *has already sent FIN*, RECEIVEs must be satisfied by text already on hand, but not yet delivered to the user. If no text is awaiting delivery, the RECEIVE will get a "error: connection closing" response. Otherwise, any remaining text can be used to satisfy the RECEIVE.
Step 3. Protocol State Machine Extraction - **Missing Transition**

Consider the transition:

```
CLOSE_WAIT --- FIN! --->
  \textcolor{red}{FALSE_\texttt{ACK}}
```

Described in the RFC as follows:

No explicit mention to \texttt{LAST_ACK}

Since the remote side \texttt{has already sent FIN}, RECEIVEs must be satisfied by text already on hand, but not yet delivered to the user. If no text is awaiting delivery, the RECEIVE will get a "error: connection closing" response. Otherwise, any remaining text can be used to satisfy the RECEIVE.
Step 3. Protocol State Machine Extraction - **Missing**

**Missing Transitions TCP**

- Gold: 2
- LinearCRF+R: 2
- NeuralCRF+R: 2

**Missing Transitions DCCP**

- Gold: 16
- LinearCRF+R: 2
- NeuralCRF+R: 2

Legend:
- Blue: Text omission or ambiguity
- Light blue: Extraction error
- Green: Prediction error
Step 3. Protocol State Machine Extraction - **Missing**

We miss *very few* transitions for **TCP**.
Step 3. Protocol State Machine Extraction - **Missing**

DCCP specifications are significantly more ambiguous.
Step 3. Protocol State Machine Extraction - **Missing**

Our **neural model** yields **fewer prediction errors** than our linear model.
Step 3. Protocol State Machine Extraction - Incorrect

Incorrect transitions for TCP are introduced due to prediction errors.
Step 3. Protocol State Machine Extraction - **Incorrect**

Another indication that the **DCCP specifications are ambiguous**
Step 3. Protocol State Machine Extraction - **Incorrect**

Our extraction method **underperforms** on DCCP, suggesting that structure is more complex.
4. Automated Attack Synthesis

RFC Specification

The nine possible states are as follows. They are listed in increasing order, so that "state => CLOSED" means the same as "state = CLOSED" or state = "OPEN" or state = "TIMEWAIT". Section 8 describes the states in more detail.

- CLOSED: Represents nonexistent connections.
- LISTEN: Represents server sockets in the passive listening state. LISTEN and CLOSED are not associated with any particular DCCP connection.
- REQUEST: A client socket enters this state, from CLOSED, after sending a DCCP-Request packet to try to initiate a connection.
- RESPOND: A server socket enters this state, from LISTEN, after receiving a DCCP-Request from a client.
- PARGOPEN: A client socket enters this state, from REQUEST, after receiving a DCCP-Response from the server. This state represents the third phase of the three-way handshake. The client may send application data in this state, but it MUST include an Acknowledgement Number on all of its packets.

FSM Interpretation

Bugs & Attacks

NLP

Attack Synthesis
4. Automated Attack Synthesis

Extracted FSM

KORG
4. Automated Attack Synthesis
4. Automated Attack Synthesis

LTL Property $\varphi$

Extracted FSM

KORG

Does $P_0 \parallel \text{vuln. channel} \parallel P_1 \models \varphi$ ?

Spin

checked
4. Automated Attack Synthesis

Extracted FSM

KORG

LTL Property $\varphi$

Does $P_0 \parallel \text{vuln. channel} \parallel P_1 \models \varphi$?

Nope. Here’s a counterexample.
4. Automated Attack Synthesis

- LTL Property $\varphi$
- Extracted FSM
- KORG
- Does $P_0 \parallel \text{vuln. channel} \parallel P_1 \models \varphi$?
- Nope. Here’s a counterexample.
- Attacker Program A
- If possible, test results against Canonical FSM
Case Studies

Transmission Control Protocol (TCP)

Datagram Congestion Control Protocol (DCCP)
Case Studies

Transmission Control Protocol (TCP)

1. No half-open connections.

2. Passive/active establishment eventually succeeds.

3. Peers don’t get stuck.

4. SYN_RECEIVED is eventually followed by ESTABLISHED, FIN_WAIT_1, or CLOSED.

Datagram Congestion Control Protocol (DCCP)
Case Studies

Transmission Control Protocol (TCP)

1. No half-open connections.
2. Passive/active establishment eventually succeeds.
3. Peers don’t get stuck.
4. SYN_RECEIVED is eventually followed by ESTABLISHED, FIN_WAIT_1, or CLOSED.

Datagram Congestion Control Protocol (DCCP)

1. The peers don’t both loop into being stuck or infinitely looping.
2. The peers are never both in TIME_WAIT.
3. The first peer doesn’t loop into being stuck or infinitely looping.
4. The peers are never both in CLOSE_REQ.
Case Studies

<table>
<thead>
<tr>
<th>TCP PROMELA program</th>
<th>Candidates Guided by $\varphi$.</th>
<th>Unconfirmed Candidates Guided by $\varphi$.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canonical</td>
<td>$\phi_1$ $\phi_2$ $\phi_3$ $\phi_4$</td>
<td>$\phi_1$ $\phi_2$ $\phi_3$ $\phi_4$</td>
</tr>
<tr>
<td>Gold</td>
<td>1 9 36 17</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>LINEARCRF+R</td>
<td>2 0 0 0</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>NEURALCRF+R</td>
<td>1 0 0 0</td>
<td>0 0 0 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DCCP PROMELA program</th>
<th>$\theta_1$ $\theta_2$ $\theta_3$ $\theta_4$</th>
<th>$\theta_1$ $\theta_2$ $\theta_3$ $\theta_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Canonical</td>
<td>0 12 0 1</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>Gold</td>
<td>0 1 0 1</td>
<td>0 0 0 0</td>
</tr>
<tr>
<td>LINEARCRF+R</td>
<td>8 2 13 1</td>
<td>2 0 13 0</td>
</tr>
<tr>
<td>NEURALCRF+R</td>
<td>5 2 9 1</td>
<td>2 0 9 0</td>
</tr>
</tbody>
</table>

- Few attacks found for TCP but all true-positives.
- Many attacks found for DCCP but some are false-positives.
- No novel attacks found.
- Attacks can be thought of as bugs. (The FSM should be resilient to attack.)
## Case Studies - Example Attacks

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Model</th>
<th>Guiding Property</th>
<th>Violated Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>NeuralCRF+R</td>
<td>1</td>
<td>1</td>
<td>Injects ACK to peer 1, causing desynchronization during establishment.</td>
</tr>
<tr>
<td>DCCP</td>
<td>LinearCRF+R</td>
<td>4</td>
<td>4</td>
<td>Spoofs each peer, guiding the other to CLOSE_REQ.</td>
</tr>
<tr>
<td>DCCP</td>
<td>NeuralCRF+R</td>
<td>2</td>
<td>4</td>
<td>Similar to 1.</td>
</tr>
</tbody>
</table>
Future Directions

- Automatically highlight omissions and ambiguities in RFC text.
- Automatically suggest bug fixes.
- Automatically extract logical properties.
- Support for secure protocols.
- RFC author in-the-loop.
- Aid RFC author in achieving unambiguous translation RFC → canonical FSM.