Abstract

This is the abstract for my master's thesis. Here is some text for formatting.

At least 18 people have been killed and dozens trapped in the Indian city of Varanasi after a flyover collapsed, crushing vehicles beneath it. The flyover was still being built when portions of its cement structure fell on the road being used under it. Officials from the National Disaster Response Force said 18 bodies had been recovered so far. A rescue operation is continuing for those believed to still be trapped, but their number and condition is unknown. Photographs and video from the scene showed cars and a bus crushed beneath the weight of the concrete, many of which still held people inside. Local media reported that a handful of people had been successfully rescued, as seven cranes attempted to lift the concrete pillar. A large crowd also gathered at the scene. One eyewitness told reporters they were nearby when the collapse happened. "At least four cars, an auto-rickshaw and a minibus were crushed under it," they said.

India's NDTV also reported that many of those trapped are believed to be construction workers who had been building the flyover. The cause of the collapse is not yet known, and an inquiry has been ordered, NDTV added. Major collapses of buildings and other infrastructure are not uncommon in India, where the enforcement of construction standards is weaker than many Western countries. In September, 33 people died when a six-storey Mumbai building toppled and more than 20 people died in 2016 when a flyover collapsed in Kolkata. Other collapses with smaller death tolls are frequent. Varanasi is the home constituency of India's Prime Minister Narendra Modi, who said he was "extremely saddened by the loss of lives due to the collapse". "I pray that the injured recover soon. Spoke to officials and asked them to ensure all possible support to those affected," he tweeted.

Copyright

Disclaimer

Acknowledgements

Table Of Contents

List of Figures

List of Tables

List of Acronyms

Chapter 1

Introduction

Some text here.[1]

Chapter 2

Background

Formal Methods

Functional Programming

Higher Order Logic (HOL) Interactive Theorem Prover

Other Interactive Theorem Provers

Chapter 3

Systems Security Engineering

3.1 NIST Special Publication 800-160

3.2 Verification & Documentation

3.3 Principle of Complete Mediation

3.3.1 Formal Verification Using Computer-Aided Reasoning

Chapter 4

Patrol Base Operations

This is the future works section. But, as I am typing this, it is the current working section for LaTeX. The point here is to get the margins in order. This means that there must be text of sufficient length to visually verify that the text meets LORI's standards. LORI is complying with SU standards for the senior thesis. Therefore, meeting LORI's standards is synonymous with meeting SU's standards. Resistance will only degrade you.

4.1 Motivation

4.2 Ranger Handbook Description

4.3 Describing The Patrol Base Operations

4.4 Hierarchy of Secure State Machines

4.4.1 OMNI-Level

4.4.2 Escape

4.4.3 Top Level

4.4.4 Horizontal Slice

4.4.4.1 ssmPlanPB

4.4.4.2 ssmMoveToORP

4.4.4.3 ssmConductORP

4.4.4.4 ssmMoveToPB

4.4.4.5 ssmConductPB

4.4.5 Vertical Slice

Chapter 5

Secure State Machine Model

5.1 State Machines

5.1.1 Next-state Function

5.1.2 Next-output Function

5.1.3 Transition Commands

5.2 Secure State Machines

5.2.1 State Machine Versus Secure State Machine

5.2.2 Transition Types

5.2.3 Authentication

5.2.4 Authorization

Chapter 6

Patrol Base Operations as Secure State Machines

6.1 ssmPB: An Example from the Hierarchy

6.1.1 Principals

6.1.2 States

6.1.3 Commands

6.1.4 Next-State Function

6.1.5 Next-Output Function

6.1.6 Authentication

6.1.7 Authorization

Chapter 7

Discussion

7.1 Recap

7.2 Mission Accomplished

7.3 Stop-Gaps, Lessons Learned, & Advice

7.4 Other Verifiable Theories

7.4.1 Platoon Theory, Soldier Theory, Squad Theory, etc.

7.4.2 Soldiers in Roles

Chapter 8

Future Work & Implications

This is the future works section. But, as I am typing this, it is the current working section for LaTeX.The point here is to get the margins in order. This means that there must be text of sufficient length to visually verify that the text meets LORI's standards. LORI is complying with SU standards for the senior thesis. Therefore, meeting LORI's standards is synonymous with meeting SU's standards. Resistance will only degrade you.

8.1 The Devil Is in The Details

Of course, there are top margins and bottom margins. This means that we'll need more text. You know, the best wait to generate text is to just cut-n-paste some random stuff. Perinton, N.Y. – The FBI conducted a search of Morgan Management LLC's offices in Monroe County Monday as part of an ongoing investigation into the development company's business practices, according to Rochester area media reports.

Agents were seen carrying boxes in and out of the company's headquarters at 1080

Pittsford Victor Road in the town of Perinton, according to the reports.

An FBI spokeswoman confirmed that agents conducted ”court-authorized activity at 1080 Pittsford Victor Road,” the Democrat & Chronicle reported. The company’s founder, developer Robert Morgan, was in the office as agents conducted the search, the newspaper said.

The newspaper reported in September that a federal investigation is focused on bank loans to Morgan’s real estate portfolio, which, according to the company’s website, has grown to 140 properties and more than 34,000 apartment units across 14 states since the company’s founding in 1979.

The investigation is centered largely on Buffalo-region apartment complexes purchased by Morgan’s companies and whether the information the company gave lenders to obtain the loans was accurate, according to the newspaper.

However, the Buffalo News reported in March that the investigation includes a look at Morgan’s purchase of the Rugby Square apartment complex on Dorchester Avenue in Syracuse. One of Morgan’s companies borrowed $5.56 million to buy the apartment complex in a distress sale in 2012, then obtained a new $9 million mortgage on the property just 10 months later after reporting a major turnaround of the complex, the newspaper said.

Morgan has said his companies have done nothing illegal to obtain financing. No charges have been filed in connection with the investigation.

According to the company’s website, Morgan operates 13 apartment complexes in the Syracuse area.

Perinton, N.Y. – The FBI conducted a search of Morgan Management LLC’s offices in Monroe County Monday as part of an ongoing investigation into the development company's business practices, according to Rochester area media reports.

Agents were seen carrying boxes in and out of the company's headquarters at 1080 Pittsford Victor Road in the town of Perinton, according to the reports.

An FBI spokeswoman confirmed that agents conducted "court-authorized activity at 1080 Pittsford Victor Road," the Democrat & Chronicle reported. The company's founder, developer Robert Morgan, was in the office as agents conducted the search, the newspaper said.

The newspaper reported in September that a federal investigation is focused on bank loans to Morgan's real estate portfolio, which, according to the company's website, has grown to 140 properties and more than 34,000 apartment units across 14 states since the company's founding in 1979.

The investigation is centered largely on Buffalo-region apartment complexes purchased by Morgan's companies and whether the information the company gave lenders to obtain the loans was accurate, according to the newspaper.

However, the Buffalo News reported in March that the investigation includes a look at Morgan's purchase of the Rugby Square apartment complex on Dorchester Avenue in Syracuse. One of Morgan's companies borrowed $5.56 million to buy the apartment complex in a distress sale in 2012, then obtained a new $9 million mortgage on the property just 10 months later after reporting a major turnaround of the complex, the newspaper said.

Morgan has said his companies have done nothing illegal to obtain financing. No charges have been filed in connection with the investigation.

According to the company's website, Morgan operates 13 apartment complexes in the Syracuse area.

Perinton, N.Y. – The FBI conducted a search of Morgan Management LLC's offices in Monroe County Monday as part of an ongoing investigation into the development company's business practices, according to Rochester area media reports.

Agents were seen carrying boxes in and out of the company's headquarters at 1080 Pittsford Victor Road in the town of Perinton, according to the reports.

An FBI spokeswoman confirmed that agents conducted "court-authorized activity at 1080 Pittsford Victor Road," the Democrat & Chronicle reported. The company's founder, developer Robert Morgan, was in the office as agents conducted the search, the newspaper said.

The newspaper reported in September that a federal investigation is focused on bank loans to Morgan's real estate portfolio, which, according to the company's website, has grown to 140 properties and more than 34,000 apartment units across 14 states since the company's founding in 1979.

The investigation is centered largely on Buffalo-region apartment complexes purchased by Morgan's companies and whether the information the company gave lenders to obtain the loans was accurate, according to the newspaper.

However, the Buffalo News reported in March that the investigation includes a look at Morgan's purchase of the Rugby Square apartment complex on Dorchester Avenue in Syracuse. One of Morgan's companies borrowed $5.56 million to buy the apartment complex in a distress sale in 2012, then obtained a new $9 million mortgage on the property just 10 months later after reporting a major turnaround of the complex, the newspaper said.

Morgan has said his companies have done nothing illegal to obtain financing. No charges have been filed in connection with the investigation.

According to the company's website, Morgan operates 13 apartment complexes in the Syracuse area.

8.2 Accountability Systems

Perinton, N.Y. – The FBI conducted a search of Morgan Management LLC's offices in Monroe County Monday as part of an ongoing investigation into the development company's business practices, according to Rochester area media reports.

Agents were seen carrying boxes in and out of the company's headquarters at 1080 Pittsford Victor Road in the town of Perinton, according to the reports.

An FBI spokeswoman confirmed that agents conducted "court-authorized activity at 1080 Pittsford Victor Road," the Democrat & Chronicle reported. The company's founder, developer Robert Morgan, was in the office as agents conducted the search, the newspaper said.

The newspaper reported in September that a federal investigation is focused on bank loans to Morgan's real estate portfolio, which, according to the company's website, has grown to 140 properties and more than 34,000 apartment units across 14 states since the company's founding in 1979.

The investigation is centered largely on Buffalo-region apartment complexes purchased by Morgan's companies and whether the information the company gave lenders to obtain the loans was accurate, according to the newspaper.

However, the Buffalo News reported in March that the investigation includes a look at Morgan's purchase of the Rugby Square apartment complex on Dorchester Avenue in Syracuse. One of Morgan's companies borrowed $5.56 million to buy the apartment complex in a distress sale in 2012, then obtained a new $9 million mortgage on the property just 10 months later after reporting a major turnaround of the complex, the newspaper said.

Morgan has said his companies have done nothing illegal to obtain financing. No charges have been filed in connection with the investigation.

According to the company's website, Morgan operates 13 apartment complexes in the Syracuse area.

8.3 Applicability

Perinton, N.Y. – The FBI conducted a search of Morgan Management LLC's offices in Monroe County Monday as part of an ongoing investigation into the development company's business practices, according to Rochester area media reports.

Agents were seen carrying boxes in and out of the company's headquarters at 1080 Pittsford Victor Road in the town of Perinton, according to the reports.

An FBI spokeswoman confirmed that agents conducted "court-authorized activity at 1080 Pittsford Victor Road," the Democrat & Chronicle reported. The company's founder, developer Robert Morgan, was in the office as agents conducted the search, the newspaper said.

The newspaper reported in September that a federal investigation is focused on bank loans to Morgan's real estate portfolio, which, according to the company's website, has grown to 140 properties and more than 34,000 apartment units across 14 states since the company's founding in 1979.

The investigation is centered largely on Buffalo-region apartment complexes purchased by Morgan's companies and whether the information the company gave lenders to obtain the loans was accurate, according to the newspaper.

However, the Buffalo News reported in March that the investigation includes a look at Morgan's purchase of the Rugby Square apartment complex on Dorchester Avenue in Syracuse. One of Morgan's companies borrowed $5.56 million to buy the apartment complex in a distress sale in 2012, then obtained a new $9 million mortgage on the property just 10 months later after reporting a major turnaround of the complex, the newspaper said.

Morgan has said his companies have done nothing illegal to obtain financing. No charges have been filed in connection with the investigation.

According to the company's website, Morgan operates 13 apartment complexes in the Syracuse area.

Appendices

Appendix A

Access Control Logic Theories: Pretty-Printed Theories

Appendix B

Secure State Machine Theories: HOL Script Files

B.1 ssm

```
(***********************************************************************
(* Secure State Machine Theory: authentication, authorization, and state
*)
(* interpretation.
*)
(* Author: Shiu-Kai Chin
*)
(* Date: 27 November 2015
*)
(***********************************************************************


structure ssmScript = struct

(* ==== Interactive mode ====
app load ["TypeBase", "ssminfRules","listTheory","optionTheory","acl_infRu
          "satListTheory","ssmTheory"];
open TypeBase listTheory ssminfRules optionTheory acl_infRules satListTheo

app load ["TypeBase", "ssminfRules","listTheory","optionTheory","acl_infRu
          "satListTheory"];
open TypeBase listTheory ssminfRules optionTheory acl_infRules satListTheo
     ssmTheory
 ==== end interactive mode ==== *)

open HolKernel boolLib Parse bossLib
open TypeBase listTheory optionTheory ssminfRules acl_infRules satListTheo
```

```
(**********************)
(* create a new theory *)
(**********************)
val _ = new_theory "ssm";

(* ------------------------------------------------------------------------
(* Define the type of transition: discard, execute, or trap. We discard fro
*)
(* the input stream those inputs that are not of the form P says command. W
*)
(* execute commands that users and supervisors are authorized for. We trap
*)
(* commands that users are not authorized to execute.
*)
(* ------------------------------------------------------------------------

(* ------------------------------------------------------------------------
(* In keeping with virtual machine design principles as described by Popek
*)
(* and Goldberg, we add a TRAP instruction to the commands by users.
*)
(* In effect, we are LIFTING the commands available to users to include the
*)
(* TRAP instruction used by the state machine to handle authorization erro
(* ------------------------------------------------------------------------

val _ =
Datatype
`trType =
  discard 'cmdlist | trap 'cmdlist | exec 'cmdlist`

val trType_distinct_clauses = distinct_of ``:'cmdlist trType``
val _ = save_thm("trType_distinct_clauses",trType_distinct_clauses)

val trType_one_one = one_one_of ``:'cmdlist trType``
val _ = save_thm("trType_one_one",trType_one_one)

(* ------------------------------------------------------------------------
(* Define configuration to include the security context within which the
*)
(* inputs are evaluated. The components are as follows: (1) the authentica
(* function, (2) the intepretation of the state, (3) the security context,
*)
(* (4) the input stream, (5) the state, and (6) the output stream.
*)
(* ------------------------------------------------------------------------
val _ =
Datatype
```

```
`configuration =
 CFG
  (('command option,'principal,'d,'e)Form -> bool)
  (('state -> ('command option,'principal,'d,'e)Form list ->
   ('command option,'principal,'d,'e)Form list))
  ((('command option,'principal,'d,'e)Form list) ->
   (('command option,'principal,'d,'e)Form list))
  (((('command option,'principal,'d,'e)Form) list) list)
  ('state)
  ('output list)`

(* ------------------------------------------------------------------------
(* Prove one-to-one properties of configuration
*)
(* ------------------------------------------------------------------------
val configuration_one_one =
    one_one_of ``:('command option,'d,'e,'output,'principal,'state)configura

val _ = save_thm("configuration_one_one",configuration_one_one)

(* ------------------------------------------------------------------------
(* The interpretation of configuration is the conjunction of the formulas 
*)
(* the context and the first element of a non-empty input stream.
*)
(* ------------------------------------------------------------------------
val CFGInterpret_def =
Define
`CFGInterpret
 ((M:('command option,'b,'principal,'d,'e)Kripke),Oi:'d po,Os:'e po)
 (CFG
  (elementTest:('command option,'principal,'d,'e)Form -> bool)
  (stateInterp:'state -> (('command option,'principal,'d,'e)Form list) ->
   (('command option,'principal,'d,'e)Form list))
  (context:(('command option,'principal,'d,'e)Form list) ->
   (('command option,'principal,'d,'e)Form list))
  ((x:('command option,'principal,'d,'e)Form list)::ins)
  (state:'state)
  (outStream:'output list))
 =
  ((M,Oi,Os) satList (context x)) /\
  ((M,Oi,Os) satList x) /\
  ((M,Oi,Os) satList (stateInterp state x))`

(***************************************************************************
(* In the following definitions of authenticationTest, extractCommand, and
*)
(* commandList, we implicitly assume that the only authenticated inputs are
*)
(* of the form P says phi, i.e., we know who is making statement phi.
*)
(*********************************************************************

val authenticationTest_def =
Define
`authenticationTest
 (elementTest:('command option,'principal,'d,'e)Form -> bool)
 (x:('command option,'principal,'d,'e)Form list) =
 FOLDR (\p q.p /\ q) T (MAP elementTest x)`;

val extractCommand_def =
Define
`extractCommand (P says (prop (SOME cmd)):('command option,'principal,'d,'e
   cmd`;

val commandList_def =
Define
`commandList (x:('command option,'principal,'d,'e)Form list) =
 MAP extractCommand x`;

val extractPropCommand_def =
Define
`(extractPropCommand (P says (prop (SOME cmd)):('command option,'principal,
   ((prop (SOME cmd)):('command option,'principal,'d,'e)Form))`;

val propCommandList_def =
Define
`propCommandList (x:('command option,'principal,'d,'e)Form list) =
 MAP extractPropCommand x`;

val extractInput_def =
Define
`extractInput (P says (prop x):('command option,'principal,'d,'e)Form) = x

val inputList_def =
Define
`inputList (xs:('command option,'principal,'d,'e)Form list) =
 MAP extractInput xs`;

(* --------------------------------------------------------------------
(* Define transition relation among configurations. This definition is
*)
(* parameterized in terms of next-state transition function and output
*)
(* function.
*)
```

```
(* ────────────────────────────────────────────────────────────────
val (TR_rules, TR_ind, TR_cases) =
Hol_reln
‘(!(elementTest:('command option,'principal,'d,'e)Form -> bool)
    (NS: 'state -> ('command option list) trType -> 'state) M Oi Os Out (s:
    (context:(('command option,'principal,'d,'e)Form list) ->
   (('command option,'principal,'d,'e)Form list))
    (stateInterp:'state -> ('command option,'principal,'d,'e)Form list ->
     ('command option,'principal,'d,'e)Form list)
    (x:('command option,'principal,'d,'e)Form list)
    (ins:('command option,'principal,'d,'e)Form list list)
    (outs:'output list).
 (authenticationTest elementTest x) /\
 (CFGInterpret (M,Oi,Os)
  (CFG elementTest stateInterp context (x::ins) s outs)) ==>
 (TR
  ((M:('command option,'b,'principal,'d,'e)Kripke),Oi:'d po,Os:'e po)
  (exec (inputList x))
  (CFG elementTest stateInterp context (x::ins) s outs)
  (CFG elementTest stateInterp context ins
       (NS s (exec (inputList x)))
       ((Out s (exec (inputList x)))::outs)))) /\
 (!(elementTest:('command option,'principal,'d,'e)Form -> bool)
    (NS: 'state -> ('command option list) trType -> 'state) M Oi Os Out (s:
    (context:(('command option,'principal,'d,'e)Form list) ->
   (('command option,'principal,'d,'e)Form list))
    (stateInterp:'state -> ('command option,'principal,'d,'e)Form list ->
     ('command option,'principal,'d,'e)Form list)
    (x:('command option,'principal,'d,'e)Form list)
    (ins:('command option,'principal,'d,'e)Form list list)
    (outs:'output list).
 (authenticationTest elementTest x)  /\
 (CFGInterpret (M,Oi,Os)
  (CFG elementTest stateInterp context (x::ins) s outs)) ==>
 (TR
  ((M:('command option,'b,'principal,'d,'e)Kripke),Oi:'d po,Os:'e po)
  (trap (inputList x))
 (CFG elementTest stateInterp context (x::ins) s outs)
 (CFG elementTest stateInterp context ins
      (NS s (trap (inputList x)))
      ((Out s (trap (inputList x)))::outs)))) /\
 (!(elementTest:('command option,'principal,'d,'e)Form -> bool)
    (NS: 'state -> ('command option list) trType -> 'state) M Oi Os Out (s:
    (context:(('command option,'principal,'d,'e)Form list) ->
   (('command option,'principal,'d,'e)Form list))
    (stateInterp:'state -> ('command option,'principal,'d,'e)Form list ->
     ('command option,'principal,'d,'e)Form list)
    (x:('command option,'principal,'d,'e)Form list)
    (ins:('command option,'principal,'d,'e)Form list list)
    (outs:'output list).
  ~(authenticationTest elementTest x) ==>
  (TR
   ((M:('command option,'b,'principal,'d,'e)Kripke),Oi:'d po,Os:'e po)
   (discard (inputList x))
  (CFG elementTest stateInterp context (x::ins) s outs)
  (CFG elementTest stateInterp context ins
       (NS s (discard (inputList x)))
       ((Out s (discard (inputList x)))::outs))))`


(* ---------------------------------------------------------------------------
(* Split up TR_rules into individual clauses
*)
(* ---------------------------------------------------------------------------
val [rule0,rule1,rule2] = CONJUNCTS TR_rules

(*****************************************************************************
(* Prove the converse of rule0, rule1, and rule2
*)
(*****************************************************************************
val TR_lemma0 =
TAC_PROOF(([],flip_TR_rules rule0),
DISCH_TAC THEN
IMP_RES_TAC TR_cases THEN
PAT_ASSUM
  ``exec cmd = y``
  (fn th => ASSUME_TAC(REWRITE_RULE[trType_one_one,trType_distinct_clauses]t
PROVE_TAC[configuration_one_one,list_11,trType_distinct_clauses])


val TR_lemma1 =
TAC_PROOF(([],flip_TR_rules rule1),
DISCH_TAC THEN
IMP_RES_TAC TR_cases THEN
PAT_ASSUM
  ``trap cmd = y``
  (fn th => ASSUME_TAC(REWRITE_RULE[trType_one_one,trType_distinct_clauses]t
 PROVE_TAC[configuration_one_one,list_11,trType_distinct_clauses])

val TR_lemma2 =
TAC_PROOF(([],flip_TR_rules rule2),
DISCH_TAC THEN
IMP_RES_TAC TR_cases THEN
PAT_ASSUM
  ``discard (inputList x)= y``
  (fn th => ASSUME_TAC(REWRITE_RULE[trType_one_one,trType_distinct_clauses]t
PROVE_TAC[configuration_one_one, list_11, trType_distinct_clauses])

val TR_rules_converse =
TAC_PROOF(([], flip_TR_rules TR_rules),
REWRITE_TAC[TR_lemma0, TR_lemma1, TR_lemma2])

val TR_EQ_rules_thm = TR_EQ_rules TR_rules TR_rules_converse

val _ = save_thm("TR_EQ_rules_thm",TR_EQ_rules_thm)

val [TRrule0,TRrule1,TR_discard_cmd_rule] = CONJUNCTS TR_EQ_rules_thm

val _ = save_thm("TRrule0",TRrule0)
val _ = save_thm("TRrule1",TRrule1)
val _ = save_thm("TR_discard_cmd_rule",TR_discard_cmd_rule)

(* ----------------------------------------------------------------------
(* If (CFGInterpret
*)
(*      (M,Oi,Os)
*)
(*      (CFG elementTest stateInterpret certList
*)
(*           ((P says (prop (CMD cmd)))::ins) s outs) ==>
*)
(*     ((M,Oi,Os) sat (prop (CMD cmd))))
*)
(* is a valid inference rule, then executing cmd the exec(CMD cmd) transiti
(* occurs if and only if prop (CMD cmd), elementTest, and
*)
(* CFGInterpret (M,Oi,Os)
*)
(*  (CFG elementTest stateInterpret certList (P says prop (CMD cmd)::ins) s
*)
(* are true.
*)
(* ----------------------------------------------------------------------
val TR_exec_cmd_rule =
TAC_PROOF(([],
``!elementTest context stateInterp (x:('command option,'principal,'d,'e)For
   ins s outs.
  (!M Oi Os.
  (CFGInterpret
   ((M :('command option, 'b, 'principal, 'd, 'e) Kripke),(Oi :'d po), (Os :
   (CFG elementTest
        (stateInterp:'state -> ('command option,'principal,'d,'e)Form list -
         ('command option,'principal,'d,'e)Form list) context
        (x::ins)
        (s:'state) (outs:'output list))) ==>
   (M,Oi,Os) satList (propCommandList (x:('command option, 'principal, 'd,
(!NS Out M Oi Os.
 TR
   ((M :('command option, 'b, 'principal, 'd, 'e) Kripke),(Oi :'d po),
    (Os :'e po)) (exec (inputList x))
   (CFG (elementTest :('command option, 'principal, 'd, 'e) Form -> bool)
             (stateInterp:'state -> ('command option,'principal,'d,'e)Form li
              ('command option,'principal,'d,'e)Form list)
             (context :('command option, 'principal, 'd, 'e) Form list ->
              ('command option, 'principal, 'd, 'e) Form list)
             (x::ins)
             (s :'state) (outs :'output list))
    (CFG elementTest stateInterp context ins
             ((NS :'state -> 'command option list trType -> 'state) s (exec (
             (Out s (exec (inputList x))::outs)) <=>
   (authenticationTest elementTest x) /\
   (CFGInterpret (M,Oi,Os)
    (CFG elementTest stateInterp context (x::ins) s outs)) /\
    (M,Oi,Os) satList (propCommandList x))``),
REWRITE_TAC[TRrule0] THEN
REPEAT STRIP_TAC THEN
EQ_TAC THEN
REPEAT STRIP_TAC THEN
PROVE_TAC[])

val _ = save_thm("TR_exec_cmd_rule",TR_exec_cmd_rule)

(* ------------------------------------------------------------------------
(* If (CFGInterpret
*)
(*       (M,Oi,Os)
*)
(*       (CFG elementTest stateInterpret certList
*)
(*             ((P says (prop (CMD cmd)))::ins) s outs) ==>
*)
(*     ((M,Oi,Os) sat (prop TRAP)))
*)
(* is a valid inference rule, then executing cmd the trap(CMD cmd) transiti
(* occurs if and only if prop TRAP, elementTest, and
*)
(* CFGInterpret (M,Oi,Os)
*)
(*  (CFG elementTest stateInterpret certList (P says prop (CMD cmd)::ins)
*)
(*         s outs) are true.
*)
```

```
(* ----------------------------------------------------------------------------
val TR_trap_cmd_rule =
TAC_PROOF(
([],``!elementTest context stateInterp (x:('command option,'principal,'d,'e
   ins s outs.
  (!M Oi Os.
  (CFGInterpret
   ((M :('command option, 'b, 'principal, 'd, 'e) Kripke),(Oi :'d po), (Os :
   (CFG elementTest
        (stateInterp:'state -> ('command option,'principal,'d,'e)Form list -
         ('command option,'principal,'d,'e)Form list) context
        (x::ins)
        (s:'state) (outs:'output list))) ==>
   (M,Oi,Os) sat (prop NONE)) ==>
 (!NS Out M Oi Os.
  TR
   ((M :('command option, 'b, 'principal, 'd, 'e) Kripke),(Oi :'d po),
    (Os :'e po)) (trap (inputList x))
   (CFG (elementTest :('command option, 'principal, 'd, 'e) Form -> bool)
            (stateInterp:'state -> ('command option,'principal,'d,'e)Form li
             ('command option,'principal,'d,'e)Form list)
            (context :('command option, 'principal, 'd, 'e) Form list ->
             ('command option, 'principal, 'd, 'e) Form list)
            (x::ins)
            (s :'state) (outs :'output list))
    (CFG elementTest stateInterp context ins
            ((NS :'state -> 'command option list trType -> 'state) s (trap (
            (Out s (trap (inputList x))::outs)) <=>
   (authenticationTest elementTest x) /\
   (CFGInterpret (M,Oi,Os)
    (CFG elementTest stateInterp context (x::ins) s outs)) /\
   (M,Oi,Os) sat (prop NONE))``),
REWRITE_TAC[TRrule1] THEN
REPEAT STRIP_TAC THEN
EQ_TAC THEN
REPEAT STRIP_TAC THEN
PROVE_TAC[])

val _ = save_thm("TR_trap_cmd_rule",TR_trap_cmd_rule)

(* ==== start here ====
==== end here ==== *)

val _ = export_theory ();
val _ = print_theory "-";

end (* structure *)
```

Appendix C

Secure State Machine Theories Applied to Patrol Base Operations: HOL Script Files

C.1 OMNILevel

```
(****************************************************************************
(* OMNIScript
*)
(* Author: Lori Pickering
*)
(* Date: 10 May 2018
*)
(* This file is intended to allow for integration among the ssms.
The idea    *)
(* is to provide an OMNI-level integrating theory, in the sense of a super-
*)
(* conscious that knows when each ssm is complete and provides that info tc
*)
(* higher-level state machines.
*)
(****************************************************************************

structure OMNIScript = struct

(* ==== Interactive Mode =====
app load ["TypeBase","listTheory", "optionTheory",
          "OMNITypeTheory",
            "acl_infRules","aclDrulesTheory","aclrulesTheory"];
open TypeBase listTheory optionTheory
      OMNITypeTheory
      acl_infRules aclDrulesTheory aclrulesTheory
 ====  End Interactive Mode ====  *)

open HolKernel Parse boolLib bossLib;
open TypeBase listTheory optionTheory
open OMNITypeTheory
open acl_infRules aclDrulesTheory aclrulesTheory

val _ = new_theory "OMNI";
(*************************************************************************
(* Define slCommands for OMNI.
*)
(*************************************************************************
(* ==== Area 52 ==========

val _=
Datatype `stateRole = Omni`

val _=
Datatype `omniCommand = ssmPlanPBComplete
                      | ssmMoveToORPComplete
                      | ssmConductORPComplete
                      | ssmMoveToPBComplete
                      | ssmConductPBComplete`

val omniCommand_distinct_clauses = distinct_of ``:omniCommand``
val _= save_thm("omniCommand_distinct_clauses",
                omniCommand_distinct_clauses)

val _=
Datatype `slCommand = OMNI omniCommand`


val omniAuthentication_def =
Define
`(omniAuthentication
        (Name Omni says prop (cmd:((slCommand command) option))
        :((slCommand command) option, stateRole, 'd,'e)Form) = T) /\
 (omniAuthentication _ = F)`

val omniAuthorization_def =
Define
`(omniAuthorization
        (Name Omni controls prop (cmd:((slCommand command) option))
        :((slCommand command) option, stateRole, 'd,'e)Form) = T) /\
 (omniAuthorization _ = F)`


This may not be necessary...But, it is interesting.  Save for a later time.
(*****************************************************************************
(* Prove that
*)
(*   Omni says omniCommand ==> omniCommand
*)
(*****************************************************************************

set_goal([],
     ``(Name Omni says prop (cmd:((slCommand command) option))
          :((slCommand command) option, stateRole, 'd,'e)Form) ==>
                         prop (cmd:((slCommand command) option))``)



val th1 = ASSUME``(Name Omni says prop (cmd:((slCommand command) option))
          :((slCommand command) option, stateRole, 'd,'e)Form) = TT``
val th2 = REWRITE_RULE[omniAuthentication_def]th1

 ========= End Area 52 ==== *)

val _= export_theory();
end
```

C.2 TopLevel

C.2.1 PBTypeIntegrated Theory: Type Definitions

```
(*****************************************************************************
(* PBTypeIntegrated
*)
(* Author: Lori Pickering
*)
(* Date 12 May 2018
*)
(* This theory contains the type definitions for ssmPBIntegrated
*)
(*****************************************************************************
structure PBTypeIntegratedScript = struct
```

```
(* ===== Interactive Mode ====
app load ["TypeBase"]
open TypeBase
 ==== end Interactive Mode ==== *)


open HolKernel Parse boolLib bossLib;
open TypeBase

val _= new_theory "PBTypeIntegrated";

(*********************************************************************
(* Define types
*)
(*********************************************************************
val _=
Datatype `plCommand = crossLD  (* Move to MOVE_TO_ORP state *)
                    | conductORP
                    | moveToPB
                    | conductPB
                    | completePB
                    | incomplete`

val plCommand_distinct_clauses = distinct_of ``:plCommand``
val _= save_thm("plCommand_distinct_clauses",
                plCommand_distinct_clauses)

val _=
Datatype `omniCommand = ssmPlanPBComplete
                      | ssmMoveToORPComplete
                      | ssmConductORPComplete
                      | ssmMoveToPBComplete
                      | ssmConductPBComplete`

val omniCommand_distinct_clauses = distinct_of ``:omniCommand``
val _= save_thm("omniCommand_distinct_clauses",
                omniCommand_distinct_clauses)


val _=
Datatype `slCommand = PL plCommand
                    | OMNI omniCommand`


val slCommand_distinct_clauses = distinct_of ``:slCommand``
val _= save_thm("slCommand_distinct_clauses",
                slCommand_distinct_clauses)
```

```
val slCommand_one_one = one_one_of ``:slCommand``
val _= save_thm("slCommand_one_one", slCommand_one_one)



val _=
Datatype `stateRole = PlatoonLeader | Omni`

val stateRole_distinct_clauses = distinct_of ``:stateRole``
val _= save_thm("stateRole_distinct_clauses",
                stateRole_distinct_clauses)


val _=
Datatype `slState = PLAN_PB
                  | MOVE_TO_ORP
                  | CONDUCT_ORP
                  | MOVE_TO_PB
                  | CONDUCT_PB
                  | COMPLETE_PB`

val slState_distinct_clauses = distinct_of ``:slState``
val _ = save_thm("slState_distinct_clauses",slState_distinct_clauses)

val _=
Datatype `slOutput = PlanPB
                   | MoveToORP
                   | ConductORP
                   | MoveToPB
                   | ConductPB
                   | CompletePB
                   | unAuthenticated
                   | unAuthorized`

val slOutput_distinct_clauses = distinct_of ``:slOutput``
val _ = save_thm("slOutput_distinct_clauses",slOutput_distinct_clauses)


val _= export_theory();
end
```

C.2.2 PBIntegratedDef Theory: Authentication & Authorization Definitions

```
(*****************************************************************************
(* PBIntegratedDefTheory
*)
(* Author: Lori Pickering
*)
(* Date: 7 May 2018
*)
(* Definitions for ssmPBIntegratedTheory.
*)
(****************************************************************************
structure PBIntegratedDefScript = struct

(* ===== Interactive Mode ====
app load  ["TypeBase", "listTheory","optionTheory",
           "uavUtilities",
          "OMNITypeTheory",
          "PBIntegratedDefTheory","PBTypeIntegratedTheory"];

open TypeBase listTheory optionTheory
     aclsemanticsTheory aclfoundationTheory
     uavUtilities
     OMNITypeTheory
     PBIntegratedDefTheory PBTypeIntegratedTheory
 ==== end Interactive Mode ==== *)

open HolKernel Parse boolLib bossLib;
open TypeBase listTheory optionTheory
open  uavUtilities
open OMNITypeTheory  PBTypeIntegratedTheory

val _ = new_theory "PBIntegratedDef";
(* ---------------------------------------------------------------------------
(* state Interpretation function
*)
(* ---------------------------------------------------------------------------
(* This function doesn't do anything but is necessary to specialize other
*)
(* theorems.
*)
(* ---------------------------------------------------------------------------
val secContext_def = Define `
    secContext (x:((slCommand command)option, stateRole, 'd,'e)Form list) =
        [(TT:((slCommand command)option, stateRole, 'd,'e)Form)]`

val secHelper =
Define`
  (secHelper (cmd:omniCommand) =
     [(Name Omni) controls prop (SOME (SLc (OMNI (cmd:omniCommand))))])`

val getOmniCommand_def =
Define‘
  (getOmniCommand ([]:((slCommand command)option, stateRole, 'd,'e)Form lis
                       = invalidOmniCommand:omniCommand) /\
  (getOmniCommand (((Name Omni) controls prop (SOME (SLc (OMNI cmd)))):: xs)
                       = (cmd:omniCommand)) /\
  (getOmniCommand ((x:((slCommand command)option, stateRole, 'd,'e)Form):: x
                       =  (getOmniCommand xs))‘

val secAuthorization_def =
Define‘
  (secAuthorization (xs:((slCommand command)option, stateRole,'d,'e)Form li
                  = secHelper (getOmniCommand xs)) ‘


val secContext_def =
Define‘
 (secContext (PLAN_PB) ((x:((slCommand command)option, stateRole, 'd,'e)For
         [(prop (SOME (SLc (OMNI (ssmPlanPBComplete))))
          :((slCommand command)option, stateRole, 'd,'e)Form) impf
          (Name PlatoonLeader) controls prop (SOME (SLc (PL crossLD)))
           :((slCommand command)option, stateRole, 'd,'e)Form])
/\
 (secContext (MOVE_TO_ORP) ((x:((slCommand command)option, stateRole, 'd,'e
         [prop (SOME (SLc  (OMNI (ssmMoveToORPComplete)))) impf
          (Name PlatoonLeader) controls prop (SOME (SLc (PL conductORP)))])
 (secContext (CONDUCT_ORP) ((x:((slCommand command)option, stateRole, 'd,'e
         [prop (SOME (SLc (OMNI (ssmConductORPComplete)))) impf
          (Name PlatoonLeader) controls prop (SOME (SLc (PL moveToPB)))]) /\
 (secContext (MOVE_TO_PB) ((x:((slCommand command)option, stateRole, 'd,'e)
         [prop (SOME (SLc (OMNI (ssmMoveToPBComplete)))) impf
          (Name PlatoonLeader) controls prop (SOME (SLc (PL conductPB)))]) /
 (secContext (CONDUCT_PB) ((x:((slCommand command)option, stateRole, 'd,'e)
         [prop (SOME (SLc (OMNI (ssmConductPBComplete)))) impf
          (Name PlatoonLeader) controls prop (SOME (SLc (PL completePB)))])‘

(* ===== Area 52 ====

 ==== End Area 52 ==== *)

val _= export_theory();
end
```

C.2.3 ssmPlanPBIntegrated Theory: Theorems

```
(**********************************************************************
(* ssmPBIntegratedTheory
*)
```

```
(* Author: Lori Pickering
*)
(* Date: 7 May 2018
*)
(* This theory aims to integrate the topLevel ssm with the sublevel ssms.
It *)
(* does this by adding a condition to the security context. In particular,
*)
(* it requires that the "COMPLETE" state in the subLevel ssm must preceede
*)
(* transition to the next state at the topLeve. I.e.,
*)
(*   planPBComplete ==>
*)
(*   PlatoonLeader controls crossLD.
*)
(* In the ssmPlanPB ssm, the last state is COMPLETE. This is reached when
(* the appropriate authority says complete and the transition is made.
*)
(* Note that following the ACL, if P says x and P controls x, then x.
*)
(* Therefore, it is not necessary for anyone to say x at the topLevel, bec
(* it is already proved at the lower level.
*)
(* However, indicating that at the topLevel remains something to workout.
*)
(*******************************************************************


structure ssmPBIntegratedScript = struct

(* ===== Interactive Mode ====
app load  ["TypeBase", "listTheory","optionTheory","listSyntax",
           "acl_infRules","aclDrulesTheory","aclrulesTheory",
           "aclsemanticsTheory", "aclfoundationTheory",
           "satListTheory","ssmTheory","ssminfRules","uavUtilities",
           "OMNITypeTheory", "PBTypeIntegratedTheory","PBIntegratedDefTheor
           "ssmPBIntegratedTheory"];

open TypeBase listTheory optionTheory listSyntax
     acl_infRules aclDrulesTheory aclrulesTheory
     aclsemanticsTheory aclfoundationTheory
     satListTheory ssmTheory ssminfRules uavUtilities
     OMNITypeTheory PBTypeIntegratedTheory PBIntegratedDefTheory
     ssmPBIntegratedTheory
 ==== end Interactive Mode ==== *)

open HolKernel Parse boolLib bossLib;
open TypeBase listTheory optionTheory
open acl_infRules aclDrulesTheory aclrulesTheory
open satListTheory ssmTheory ssminfRules uavUtilities
open OMNITypeTheory PBTypeIntegratedTheory PBIntegratedDefTheory


val _ = new_theory "ssmPBIntegrated";


(*****************************************************************
(* Define next-state and next-output functions
*)
(*****************************************************************
val PBNS_def =
Define `
(PBNS PLAN_PB      (exec [SOME (SLc (PL crossLD))])     = MOVE_TO_ORP) /\
(PBNS MOVE_TO_ORP (exec [SOME (SLc (PL conductORP))]) = CONDUCT_ORP) /\
(PBNS CONDUCT_ORP (exec [SOME (SLc (PL moveToPB))])    = MOVE_TO_PB)
/\
(PBNS MOVE_TO_PB  (exec [SOME (SLc (PL conductPB))]) =  CONDUCT_PB)
/\
(PBNS CONDUCT_PB  (exec [SOME (SLc (PL completePB))]) = COMPLETE_PB) /\
(PBNS (s:slState) (trap _)     = s) /\
(PBNS (s:slState) (discard _) = s)`

val PBOut_def =
Define `
(PBOut PLAN_PB      (exec [SOME (SLc (PL crossLD))])     = MoveToORP) /\
(PBOut MOVE_TO_ORP (exec [SOME (SLc (PL conductORP))]) = ConductORP) /\
(PBOut CONDUCT_ORP (exec [SOME (SLc (PL moveToPB))])    = MoveToPB)
/\
(PBOut MOVE_TO_PB  (exec [SOME (SLc (PL conductPB))]) =  ConductPB)
/\
(PBOut CONDUCT_PB  (exec [SOME (SLc (PL completePB))]) = CompletePB) /\
(PBOut (s:slState) (trap _)     = unAuthorized) /\
(PBOut (s:slState) (discard _) = unAuthenticated)`

(*****************************************************************
(* Define authentication function
*)
(*****************************************************************
val inputOK_def =
Define `
(inputOK (((Name PlatoonLeader) says prop (cmd:((slCommand command)option))
             :((slCommand command)option, stateRole,'d,'e)Form) = T) /\
(inputOK (((Name Omni)           says prop (cmd:((slCommand command)option))
             :((slCommand command)option, stateRole,'d,'e)Form) = T) /\
(inputOK _ = F)`
```

```
(*************************************************************************
(* Prove that commands are rejected unless that are requested by a properly
*)
(* authenticated principal.
*)
(*************************************************************************

val inputOK_cmd_reject_lemma =
Q.prove(`!cmd. ~(inputOK
                  ((prop (SOME cmd))))`,
                              (PROVE_TAC[inputOK_def]))



(* ===== Just playing around with this ====
val inputOK_not_reject_lemma =
Q.prove(`!cmd.
        ~(
          (inputOK (((Name PlatoonLeader) says prop (cmd:((slCommand comma
           :((slCommand command)option, stateRole, 'd, 'e)Form)) \/
          (inputOK (((Name Omni)          says prop (cmd:((slCommand comma
           :((slCommand command)option, stateRole, 'd, 'e)Form)))

 ==== OK, done fooling around ==== *)


val _ = export_theory();

end
```

C.3 Horizontal Slice

C.3.1 ssmPlanPB

C.3.1.1 PlanPBType Theory: Type Definitions

C.3.1.2 PlanPBDef Theory: Authentication & Authorization Definitions

C.3.1.3 ssmPlanPB Theory: Theorems

C.3.2 ssmMoveToORP

C.3.2.1 MoveToORPType Theory: Type Definitions

C.3.2.2 MoveToORPDef Theory: Authentication & Authorization Definitions

C.3.2.3 ssmMoveToORP Theory: Theorems

C.3.3 ssmConductORP

C.3.3.1 ConductORPType Theory: Type Definitions

C.3.3.2 ConductORPDef Theory: Authentication & Authorization Definitions

C.3.3.3 ssmConductORP Theory: Theorems

C.3.4 ssmMoveToPB

C.3.4.1 MoveToPBType Theory: Type Definitions

C.3.4.2 MoveToPBDef Theory: Authentication & Authorization Definitions

C.3.4.3 ssmMoveToPB Theory: Theorems

C.3.5 ssmConductPB

Appendix D

Patrol Base Operations: Pretty-Printed Theories

Appendix E

Map of The File Folder Structure

References

[1] Shiu-Kai Chin and Susan Beth Older. *Access Control, Security, and Trust: A Logical Approach.* Chapman & Hall: CRC Cryptography and Network Security Series. Chapman and Hall/CRC, July 2010.