<부록2. 보안정책모델 문서>

CHAOS ver1.0
보안정책모델 문서 ver1.0
(ADV_SPM.1)

- 고등급 보안 마이크로커널 개발 -

<목 차>

1. 문서의 목적

1.1. 본 문서와 공통평가기준의 연관성

본 문서는 TOE에 해당하는 CHAOS(Chibi-based High Assurance Operating System)의 보안목표명세서 내 기술된 보안기능 요구사항을 정형화하여 모델링한 보안정책모델 문서(ADV_SPM.1)이다. ADV_SPM.1 보증요구사항은 TSF가 구현하는 보안요구사항을 정형화된 방식으로 모델링하여 표현하는 것으로 IT 제품의 설계, 구현, 검토 과정 전반에 걸쳐서 정확한 설계 및 구현에 대한 가이드라인으로 사용될 수 있다. 또한, 정형화된 수학적 모델링 방법을 사용하여 반례에 대한 보안 요구사항을 추가하는 것으로 예상치 못한 결과나 미세한 결함을 방지할 수 있다.

ADV_SPM.1 요구사항은 CC의 EAL6 등급과 EAL7에서 요구된다. 본 문서의 TOE인 CHAOS는 EAL6 등급을 달성하고 이후 EAL7 등급을 달성하고자 하기에 본 문서를 작성해야 한다.

1.2. 본 문서 작성의 상세 기준

SPM(Security Policy Modeling)문서는 다음과 같은 4가지 증거 요구사항을 만족해야 한다.

- 모델은 요구되는 경우 설명문이 지원되는 정형화된 방식이어야 하며, 모델링된 TSF의 보안정책을 식별해야 한다.
- 모델링된 모든 정책에 대해 모델은 TOE에 대한 보안성을 정의하고, TOE가 안전하지 않은 상태가 될 수 없다는 정형화된 증거를 제공해야 한다.
- 모델과 기능명세 간의 일치성은 정확한 정형화 수준이어야 한다.
- 일치성은 기능명세가 모델에 대하여 일관성 있고 완전함을 보여야한다.

정형화된 모델링 방식은 다양하게 존재하며, 이러한 모델링을 위한 언어를 정형언어, 해당 정형언어를 사용하여 요구사항을 모델링하는 것을 정형명세, 정형명세의 모순을 검증하는 행위를 정형검증, 정형명세와 정형검증을 합쳐 지원하는 프레임워크를 정형기법이라고 부른다. 평가자가 모든 정형언어를 알 수 없기 때문에 개발자는 평가자가 요청할 시 자신이 사용한 정형언어에 대한 설명서를 같이 제공할 수 있어야 한다.

개발자는 TOE에 구현된 TSF가 안전하지 않은 상태에 도달할 수 없다는 것을 입증하기 위해 정형기법을 통해 TSF를 정형명세하고 정형검증 하는 것으로 모순이 발생하지 않는다는 것을 입증해야 한다. 또한 이때 TSF 모델과 기능명세가 일관성 있고 기능명세가 TSF 모델을 완전히 커버하고 있음을 보이는 것으로 정형명세에 허점이 없음을 입증해야 한다.

1.3. 활용한 정형 명세 언어

1.3.1. Isabelle/HOL

공통평가기준에서는 정형 명세 및 검증 수행 시 Isabelle/HOL 사용을 권고한다. Isabelle은 정형 언어에 표현된 수학적 공식을 논리 연산으로 증명하기 위한 증명기이다. Isabelle은 1차 논리, 고차원 논리 (HOL: High Order Logic) 또는 Zermelo-Fraenkel 집합 이론 (ZFC)과 같은 객체 논리를 인코딩하는 데 사용되는 메타 논리 (약한 유형 이론)를 제공하는데 그 중 가장 널리 사용되는 논리 증명 방식이 Isabelle/HOL이다.

2. 보안정책모델

2.1. 개요

이 장에서는 TOE의 보안 정책에 대한 정보가 포함된 보안정책모델에 대해 제시한다. 보안정책모델은 TOE의 보안 정책을 실행하기 위해 필요한 데이터와 시스템 모델로 구성되어 있다. 데이터 모델은 시스템 요구사항을 표현하기 위한 데이터의 상세 정보이다. 데이터 모델에는 데이터 유형, 유효 범위에 대한 정보가 포함된다. 시스템 모델은 앞서 언급한 데이터 모델을 활용한 시스템 제어 관련 상세 정보이다. 시스템 모델에는 TOE 내 보안 기능을 실행하기 위해 충족해야 하는 전제 조건이 포함된다. 이후 2.2절 부터는 TOE의 보안정책모델을 공통평가기준에서 제시하는 클래스 별로 구분지어 설명한다.

2.2. 감사 관련 보안정책모델

2.2.1. 데이터 모델

감사 관련 보안정책을 수행하기 위해 필요한 데이터 모델은 아래와 같다. 저장되는 감사기록은 ‘message’라는 유형을 가지며 2가지 필드로 이루어진다. ‘identifier’는 감사기록을 발생시킨 스레드의 식별자를 의미하고 ‘behavior’는 해당 스레드로 인해 발생한 감사사건을 의미한다. 감사사건은 ‘audit_content’이라는 유형을 가지며, 값은 종류에 따라 ‘init’, ‘audit_event_generate’, ‘audit_event_generation_failed’, ‘audit_event_added_on_memory’, ‘log_storage_completely_full’, ‘log_stroage_85_percent_full’ 총 6가지 값이 부여될 수 있다.

```
datatype audit_content = init
  |audit_event_generated
  |audit_event_generation_failed
  |audit_event_added_on_memory
  |log_storage_completely_full
  |log_stroage_85_percent_full
  |thread_created

record message =
  identifier::nat
  behavior::audit_content
```

감사기록들이 저장되는 증적소에 대한 모델은 아래와 같다. 해당 모델은 상기 언급한 감사기록들이 증적됨에 따라 활용할 수 있는 메모리 양을 의미하며, ‘memory_index’, ‘memory_base’, memory_end’, ‘memory_alarm_checkpoint’ 총 4가지 필드로 이루어진다.

2.2.2. 시스템 모델

상기 데이터 모델을 활용하여 TOE가 감사 관련 보안정책을 수행하기 위해 필요한 시스템 모델은 아래와 같다. 시스템이 처음 실행되어 감사 증적소가 초기화 되는 경우 ‘init_storage’에 따라 메모리 상태가 결정된다. 이후 감사사건이 발생하는 경우 ‘audit_increment’를 통해 관련 감사기록이 증적된다. 이후 증적소의 메모리

잔여량이 0%인 경우, ‘audit_delete’에 따라 가장 오래된 감사기록이 삭제되고 이후 ‘audit_increment’에 따라 증적된다. 증적해야 하는 감사기록은 ‘audit_generate’에 따라 생성된다. 이때 악의적인 목적을 가지는 공격자로부터 인가되지 않은 감사기록이 생성되는 것을 방지하기 위해 ‘audit_generate’ 기능 수행 시, 감사사건과 관련 스레드의 값이 유효한지 점검한다.

```
definition init_storage::storage where
"init_storage ≡ ⦇memory_index = 0, memory_base = 0, memory_end = 100
                , memory_alarm_checkpoint = 85⦈"

primrec audit_increment :: "string⇒Memory⇒Memory" where
  "audit_increment x (sQueue xs ys) = sQueue (x # xs) ys"

fun audit_delete :: "Memory⇒string × Memory" where
  "audit_delete (sQueue [] []) = ([], sQueue [] [])"
| "audit_delete (sQueue xs (y # ys)) = (y, sQueue xs ys)"
| "audit_delete (sQueue xs []) = (case rev xs of y # ys ⇒ (y, sQueue [] ys))"

definition audit_generate::"audit_content⇒thread_id⇒message" where
  "audit_generate event thread ≡ if(event = audit_event_generated ∧ thread ≥ 11 ∧  thread ≤ 999)
  then ⦇identifier = thread, behavior = audit_event_generated⦈
  else if (event = log_stroage_85_percent_full ∧ thread ≥ 11 ∧  thread ≤ 999)
  then ⦇identifier = thread, behavior = log_stroage_85_percent_full⦈
  else if (event = log_storage_completely_full ∧ thread ≥ 11 ∧  thread ≤ 999)
  then ⦇identifier = thread, behavior = log_storage_completely_full⦈
  else ⦇identifier = thread, behavior = audit_event_generation_failed⦈"
```

2.2.3. 검증결과

① 최초 논리식

```
datatype audit_content = init
  |audit_event_generated
  |audit_event_generation_failed
  |audit_event_added_on_memory
  |log_storage_completely_full
  |log_stroage_85_percent_full
  |thread_created

record message =
  identifier::nat
  behavior::audit_content

definition init_storage::storage where
"init_storage ≡ ⦇memory_index = 0, memory_base = 0, memory_end = 100
                , memory_alarm_checkpoint = 85⦈"

primrec audit_increment :: "string⇒Memory⇒Memory" where
  "audit_increment x (sQueue xs ys) = sQueue (x # xs) ys"

fun audit_delete :: "Memory⇒string × Memory" where
  "audit_delete (sQueue [] []) = ([], sQueue [] [])"
| "audit_delete (sQueue xs (y # ys)) = (y, sQueue xs ys)"
| "audit_delete (sQueue xs []) = (case rev xs of y # ys ⇒ (y, sQueue [] ys))"

definition audit_generate::"audit_content⇒thread_id⇒message" where
  "audit_generate event thread ≡ if(event = audit_event_generated ∧ thread ≥ 11 ∧  thread ≤ 999)
  then ⦇identifier = thread, behavior = audit_event_generated⦈
  else if (event = log_stroage_85_percent_full ∧ thread ≥ 11 ∧  thread ≤ 999)
  then ⦇identifier = thread, behavior = log_stroage_85_percent_full⦈
  else if (event = log_storage_completely_full ∧ thread ≥ 11 ∧  thread ≤ 999)
  then ⦇identifier = thread, behavior = log_storage_completely_full⦈
  else ⦇identifier = thread, behavior = audit_event_generation_failed⦈"
```

② 결과

consts init_storage :: “storage “

consts audit_increment :: “char list ⇒ Memory ⇒ Memory “

consts audit_delete :: “Memory ⇒ char list × Memory“ Found termination order: “{} “

consts audit_generate :: “audit_content ⇒ nat ⇒ message“

2.3. 자체보호 관련 보안정책모델

본 보안정책모델은 외부로부터 시스템 제어 흐름이 위·변조되는 것을 방지하기 위해 필요하다. 보안정책모델에 따라 TOE는 제어 흐름이 미리 정의된 메모리 주소가 아닌 곳에 도달할 경우, 안전하지 않은 상태라고 판단하고 이전 실행 흐름으로 돌아간다. 이를 바탕으로 TOE는 항상 안전한 상태를 유지할 수 있다.

2.3.1. 데이터 모델

본 보안정책모델은 외부로부터 시스템 제어 흐름이 위·변조되는 것을 방지하기 위해 필요한 데이터 모델은 아래와 같다. 시스템 제어 흐름은 'Control_Flow_point' 데이터 값을 가질 수 있고, 해당 데이터 실행 시점은 리스트 형태로 'Control_flow_graph'로 관리된다. 이때 각 실행 시점에 실행되는 값이 저장된 메모리 주소는 'Functional_address'에 저장된다. 그리고 해당 메모리 주소는 리스트 형태로 'Proper_execution'에 저장되어 관리된다.

```
datatype Control_Flow_point = Executed_CFG | Unexecuted_CFG

type_synonym Control_flow_graph = "Control_Flow_point list"
type_synonym Functional_address = nat
type_synonym Execution_order = "Functional_address × Functional_address"
type_synonym Proper_execution = "Execution_order list"
```

2.3.2. 시스템 모델

상기 데이터 모델을 활용하여 TOE가 자체보호 관련 보안정책을 수행하기 위해 필요한 시스템 모델은 아래와 같다. 제어 흐름이 저장된 값이 유효한지 'Proper Roots'를 통해 점검한다. 이후 제어 흐름에 따라 TOE가 실행될 때, 임의의 제어 시점 'a'에서 'b'로 제어 흐름이 유효한지 점검하기 위해 'Proper_Edges'와 'execution_check'를 활용한다. 'Proper_edge'는 제어 흐름이 전이될 때, 사전에 정의된대로 전이가 발생하는 것인지 점검하기 위한 모델이고, 'execution_check'는 실행된 제어 흐름이 다시 반복되는 것인지 점검하기 위한 모델이다. 그리고 'Reach' 시스템 제어 흐름이 모두 도달 가능한 상태인지 점검하기 위해 'Proper_execution' 리스트의 각 세부 경로를 점검하는 모델이다.

```
definition Proper_Roots :: "Control_flow_graph ⇒ bool" where
  "Proper_Roots M ≡  Roots ≠ {} ∧ Roots ⊆ {i. i<length M}"

definition Proper_Edges :: "(Control_flow_graph × Proper_execution)⇒ bool" where
  "Proper_Edges ≡ (λ(M,E). ∀i<length E. fst(E!i)<length M ∧ snd(E!i)<length M)"

definition execution_check :: "(Execution_order × Control_flow_graph)⇒ bool" where
  "execution_check ≡ (λ(e,M). (M!fst e)=Executed_CFG ∧ (M!snd e) ≠ Executed_CFG)"

definition Reach :: "Proper_execution ⇒ nat set" where
  "Reach E ≡ {x. (∃path. 1<length path ∧ path!(length path - 1)∈Roots ∧ x=path!0
               ∧ (∀i<length path - 1. (∃j<length E. E!j=(path!(i+1), path!i))))
              ∨ x∈ Roots}"
```

2.3.3. 검증 결과

① 최초 논리식

```
datatype Control_Flow_point = Executed_CFG | Unexecuted_CFG

type_synonym Control_flow_graph = "Control_Flow_point list"
type_synonym Functional_address = nat
type_synonym Execution_order = "Functional_address × Functional_address"
type_synonym Proper_execution = "Execution_order list"

consts Roots :: "nat set"

definition Proper_Roots :: "Control_flow_graph ⇒ bool" where
  "Proper_Roots M ≡  Roots ≠ {} ∧ Roots ⊆ {i. i<length M}"

definition Proper_Edges :: "(Control_flow_graph × Proper_execution)⇒ bool" where
  "Proper_Edges ≡ (λ(M,E). ∀i<length E. fst(E!i)<length M ∧ snd(E!i)<length M)"

definition execution_check :: "(Execution_order × Control_flow_graph)⇒ bool" where
  "execution_check ≡ (λ(e,M). (M!fst e)=Executed_CFG ∧ (M!snd e) ≠ Executed_CFG)"

definition Reach :: "Proper_execution ⇒ nat set" where
  "Reach E ≡ {x. (∃path. 1<length path ∧ path!(length path - 1)∈Roots ∧ x=path!0
               ∧ (∀i<length path - 1. (∃j<length E. E!j=(path!(i+1), path!i))))
              ∨ x∈ Roots}"
```

② 결과

consts Proper_Roots :: “Control_Flow_point list ⇒ bool“

consts Proper_Edges :: “Control_Flow_point list x (nat x nat) list ⇒ bool“

consts execution_check :: “(nat x nat) ⇒ Control_Flow_point list ⇒ bool“

consts Reach :: “(nat x nat) list ⇒ nat set“

2.4. 자원활용 관련 보안정책모델

본 보안정책모델은 TOE가 항상 서비스를 제공하는 것을 보장하기 위해 필요하다. TOE는 보안정책모델에 따라 메모리와 CPU를 실행 우선순위에 따라 항상 제공할 수 있다.

2.4.1. 데이터 모델

본 보안정책모델에서 필요한 데이터 모델은 아래와 같다. ‘Thread_id’는 보안정책모델에 따라 TOE가 서비스를 제공하는 주체를 의미한다. ‘Execution_count’와 ‘Thread_total_execution_time’는 CPU 할당 시, 실행 우선순위를 고려하기 위한 데이터로 스레드가 CPU를 할당받은 횟수와 그로 인한 실행 시간을 각각 의미한다. ‘Worst_Execution_time’은 스레드의 실행 데드라인 시간을 의미한다. 스레드가 종료되지 않은 채 해당 값 이상으로 실행되었을 경우, TOE가 탑재된 전체 시스템이나 TOE의 가용성에 영향을 줄 수 있기 때문에 해당 값을 활용하여 TOE 동작을 점검한다.

```
type_synonym Thread_id = nat
type_synonym Execution_count = nat
type_synonym Worst_Execution_time = nat
type_synonym Thread_total_execution_time = nat
type_synonym memory_address = nat
```

2.4.2. 시스템 모델

상기 데이터 모델을 활용하여 TOE가 자원활용 관련 보안정책을 수행하기 위해 필요한 시스템 모델은 아래와 같다.

‘set_memory’는 스레드에게 할당될 수 있는 메모리 범위를 의미한다. ‘base_address’는 스레드에게 할당된 메모리의 크기가 가변적일 경우, 이를 계산하기 위해 선언한 값을 의미한다. ‘memory_usage_index_increment’와 ‘memory_usage_index_decrement’는 스레드에 할당된 메모리의 범위를 점검하는데 활용된다. 이때, 스레드에 할당된 메모리 크기가 ‘set_memory’에서 선언한 범위를 벗어나는 경우, 각각 ‘True’, ‘False’를 반환한다. ‘end_address’는 TOE의 전체 메모리 크기를 의미하며, 이는 ‘remain_space’를 통해 시스템에서 제공할 수 있는 메모리의 잔여량을 계산하는데 활용된다. 상기 조건들은 스레드 생성 및 삭제와 같이 메모리 할당 및 해제 오퍼레이션이 수행될 경우 점검되며 이에 대한 세부 보안정책모델은 ‘thread_create_delete’이다.

스레드가 CPU를 독점하는 것을 방지하기 위해 본 보안정책모델은 스레드 실행시간을 점검한다. ‘init_execution_time’은 스레드 생성 시, 스레드 실행 시간 값을 ‘0’으로 초기화하는 것을 의미한다. ‘thread_execute’는 스레드가 스케줄링 정책에 따라 CPU를 할당받아 실행되는 경우, 해당 값을 4씩 증가시킨다. ‘execution_time_check’는 스레드 실행 시간이 앞서 언급한 ‘thread_worst_execution_time’을 초과하는지 여부를 점검한다.

```
definition set_memory::"memory_address × memory_address" where
"set_memory = (0, 2^32-1)"

definition base_address:: "memory_address" where
  "base_address = 0"

definition memory_usage_index_increment::"memory_address⇒memory_address" where
  "memory_usage_index_increment requested_size = base_address + requested_size"

definition memory_usage_index_decrement::"memory_address⇒memory_address" where
  "memory_usage_index_decrement requested_size = base_address - requested_size"

definition boundary_check::"memory_address⇒bool" where
  "boundary_check address = (if (memory_usage_index_increment address ≥ snd set_memory) ∧
   (memory_usage_index_decrement address ≤ fst set_memory) then True else False)"

definition end_address:: "memory_address" where
  "end_address = 2^1024-1"

definition remain_space::"memory_address" where
  "remain_space = end_address - base_address"

definition remain_memory_check::"memory_address⇒bool" where
"remain_memory_check requested_size  = (if requested_size≤remain_space then True else False)"

definition thread_create_delete::"memory_address⇒memory_address⇒bool" where
  "thread_create_delete address thread_size ≡ (if boundary_check address then True else False)
   ∧ (if remain_memory_check thread_size then True else False)"
definition init_execution_time::"Thread_total_execution_time" where
  "init_execution_time = 0"

definition set_thread_worst_execution_time::"Thread_total_execution_time" where
  "set_thread_worst_execution_time = 1000"

definition thread_execute::"Thread_total_execution_time" where
  "thread_execute = init_execution_time + 4"

definition execution_time_check::"bool" where
  "execution_time_check = (if thread_execute≤set_thread_worst_execution_time
   then True else False)"
```

2.4.3. 검증 결과

① 최초 논리식

```
definition set_memory::"memory_address × memory_address" where
"set_memory = (0, 2^32-1)"

definition base_address:: "memory_address" where
  "base_address = 0"

definition memory_usage_index_increment::"memory_address⇒memory_address" where
  "memory_usage_index_increment requested_size = base_address + requested_size"

definition memory_usage_index_decrement::"memory_address⇒memory_address" where
  "memory_usage_index_decrement requested_size = base_address - requested_size"

definition boundary_check::"memory_address⇒bool" where
  "boundary_check address = (if (memory_usage_index_increment address ≥ snd set_memory) ∧
   (memory_usage_index_decrement address ≤ fst set_memory) then True else False)"

definition end_address:: "memory_address" where
  "end_address = 2^1024-1"

definition remain_space::"memory_address" where
  "remain_space = end_address - base_address"

definition remain_memory_check::"memory_address⇒bool" where
"remain_memory_check requested_size  = (if requested_size≤remain_space then True else False)"

definition thread_create_delete::"memory_address⇒memory_address⇒bool" where
  "thread_create_delete address thread_size ≡ (if boundary_check address then True else False
   ∧ (if remain_memory_check thread_size then True else False)"
```

```
definition init_execution_time::"Thread_total_execution_time" where
  "init_execution_time = 0"

definition set_thread_worst_execution_time::"Thread_total_execution_time" where
  "set_thread_worst_execution_time = 1000"

definition thread_execute::"Thread_total_execution_time" where
  "thread_execute = init_execution_time + 4"

definition execution_time_check::"bool" where
  "execution_time_check = (if thread_execute≤set_thread_worst_execution_time
   then True else False)"
```

② 결과

consts set_memory :: “nat × nat “

consts base_address :: “nat “

consts memory_usage_index_increment :: “nat ⇒ nat “

consts memory_usage_index_decrement :: “nat ⇒ nat “

consts boundary_check :: “nat ⇒ bool “

consts end_address :: “nat “

consts remain_space :: “nat “

consts remain_memory_check :: “nat ⇒ bool “

consts thread_create_delete :: “nat ⇒ nat ⇒ bool “

consts init_execution_time :: “nat “

consts set_thread_worst_execution_time :: “nat “

consts thread_execute :: “nat “

consts execution_time_check :: “bool“

2.5. 정보흐름통제 관련 보안정책모델

본 보안정책모델은 TOE의 정보의 흐름을 통제하기 위해 필요하다. TOE는 보안정책모델에 따라 정보에 접근하는 주체와 데이터 등의 접근을 허가 혹은 제한할 수 있다.

2.5.1. 데이터 모델

정보흐름통제 관련 보안정책을 수행하기 위해 필요한 데이터 모델은 아래와 같다. 저장되는 정보흐름통제 관련 기록은 'security_attribute'라는 유형을 가지며 4가지 필드로 이루어진다. 'subjectId'는 정보흐름의 주체의 ID를 의미하고, 'objectId'는 정보흐름의 객체의 ID를 의미한다. 'permission'은 해당 데이터에 대한 읽기/쓰기/생성/상속 등의 권한을 의미하며, 'data'는 정보흐름에 해당하는 데이터를 의미한다. 'security_profile'은 'security_attribute'유형의 셋을 의미한다. 'permission'은 자연수 타입으로 작성될 수 있으며, 'permissions'는 'READ', 'WRITE', 'CREATE', 'GRANT' 총 4가지 값이 부여될 수 있다. 'Id', 'subjectid', 'objectid', 'permission', 'instruction', 'data', 'IPC_message' 타입은 각각 자연수를 다른 방식으로 표현한 것이다.

```
record security_attribute=
subjectId:: int
objectId::int
permission::int
data::int

type_synonym security_profile = "(security_attribute)set"
type_synonym permission = nat


datatype permissions
= READ permission
| WRITE permission
| CREATE permission
| GRANT permission

type_synonym Id = nat
type_synonym subjectid = Id
type_synonym objectid = Id
type_synonym permission = nat
type_synonym instruction = nat
type_synonym data = nat
type_synonym IPC_message =
 "(subjectid*objectid*permission*data)"
consts IPC_message
 :: "(subjectid*objectid*permission*data)"
```

2.5.2. 시스템 모델

상기 데이터 모델을 활용하여 TOE가 정보흐름통제 관련 보안정책을 수행하기 위해 필요한 시스템 모델은 아래와 같다. 'kernel_SP'와 'valOfSecurityProfile'는 단일계층의 보안속성 관리를 의미한다. 'kernel_SP'는 커널이 가질 보안속성 값의 초기값을 의미한다. 'valOfSecurityProfile'을 통해 보안속성 값이 커널과 같은지 확인한다.

```
definition kernel_SP:: security_attribute where
"kernel_SP≡ ⦇subjectId=1, objectId=2, permission=3, data=0⦈ "

definition valOfSecurityProfile :: "security_attribute ⇒ bool"
where
"valOfSecurityProfile subjectProfile  =
(if (subjectProfile=kernel_SP)
then True
else False)"
```

'VerifyDataflow'는 접근 주체와 데이터를 입력받아서 데이터에 접근한 권한이 있는지 확인한다.

```
definition VerifyDataflow :: "subjectid ⇒ data ⇒ bool"
  where
  "VerifyDataflow subjectid data
  =
  (if subjectid ∈ subject_verification_list
  then (if data ≤ valid_bufsize
    then True
    else False)
  else False)"
```

'ControlAccess'는 접근 주체가 데이터에 접근하는 권한을 확인하고, 다음 명령문을 실행하도록 한다.

```
definition ControlAccess :: "subjectid ⇒ data ⇒ instruction"
  where
  "ControlAccess subjectid data
  =
  (if VerifyDataflow subjectid data
  then next_instruction
  else error_instruction)"
```

'CryptoBufClear'는 부분적 잔여 정보보호의 일부분으로 암호화 버퍼의 값을 제거한다. 'CMDbufClear'는 부분적 잔여 정보보호의 일부분으로 명령어 버퍼와 관련된 값을 제거한다.

```
definition CryptoBufClear :: "'a list ⇒ 'a list"
  where "CryptoBufClear buf = []"

definition CMDbufClear :: "'a list ⇒ 'a list"
  where "CMDbufClear buf = []"
```

2.5.3. 검증결과

① 최초 논리식

```
type_synonym Id = nat
type_synonym subjectid = Id
type_synonym objectid = Id
type_synonym permission = nat
type_synonym instruction = nat
type_synonym data = nat
type_synonym IPC_message =
 "(subjectid*objectid*permission*data)"
consts IPC_message
 :: "(subjectid*objectid*permission*data)"
```

```
record security_attribute=
subjectId:: int
objectId::int
permission::int
data::int

type_synonym security_profile = "(security_attribute)set"
type_synonym permission = nat


datatype permissions
= READ permission
| WRITE permission
| CREATE permission
| GRANT permission

definition kernel_SP:: security_attribute where
"kernel_SP≡ ⦇subjectId=1, objectId=2, permission=3, data=0⦈ "

definition valOfSecurityProfile :: "security_attribute ⇒ bool"
where
"valOfSecurityProfile subjectProfile  =
(if (subjectProfile=kernel_SP)
then True
else False)"

definition VerifyDataflow :: "subjectid ⇒ data ⇒ bool"
  where
  "VerifyDataflow subjectid data
  =
  (if subjectid ∈ subject_verification_list
  then (if data ≤ valid_bufsize
    then True
    else False)
  else False)"

definition ControlAccess :: "subjectid ⇒ data ⇒ instruction"
  where
  "ControlAccess subjectid data
  =
  (if VerifyDataflow subjectid data
  then next_instruction
  else error_instruction)"

definition CryptoBufClear :: "'a list ⇒ 'a list"
  where "CryptoBufClear buf = []"

definition CMDbufClear :: "'a list ⇒ 'a list"
  where "CMDbufClear buf = []"
```

② 결과

consts kernel_SP :: “security_attribute”

consts valOfSecurityProfile :: “security_attribute ⇒ bool”

consts VerifyDataflow :: “nat ⇒ nat ⇒ bool”

consts ControlAccess :: “nat ⇒ nat ⇒ nat”

consts CryptoBufClear :: “’ a list ⇒ ’ a list”

consts CMDbufClear :: “’ a list ⇒ ’ a list”

2.6. 식별 및 인증 관련 보안정책모델

본 보안정책모델은 TOE가 프로세스 등을 식별 및 인증하기 위해 필요하다. TOE는 보안정책모델에 따라 프로세스를 식별하여 허가된 명령어만 수행할 수 있도록 강제할 수 있다.

2.6.1. 데이터 모델

본 보안정책모델에서 필요한 데이터 모델은 아래와 같다. 커널이 사용자를 식별하기 위한 식별자로 'ID'를 저장하며, 각 'ID'에 해당하는 비밀 값인 'SecretValue'를 저장한다. 각 사용자의 'ID'와 해당 'ID'의 비밀 값을 쌍으로 묶어 'IDSV' 형태로 저장하며, 이를 다시 리스트로 묶은 'IDSV_list_t'를 통해 모든 사용자의 식별자와 비밀 값을 저장한다. 로그의 발생 시점은 'timestamp'로 저장되며, 문자열 형태의 로그를 'Log'에 저장한다. 여러 개의 로그를 저장하기 위해 리스트 형태로 로그를 저장하는 'Logs'를 사용하며, 이용 금지된 사용자 목록은 'BanList'로 저장되어 관리된다.

```
type_synonym ID = nat
type_synonym SecretValue = nat

type_synonym IDSV = "ID × SecretValue"
type_synonym IDSV_list_t = "IDSV list"

definition IDSV_list :: "IDSV_list_t" where "IDSV_list=[]"

datatype timestamp = nat
type_synonym Log = string
type_synonym Logs = "Log list"
type_synonym BanList = "IDSV_list_t"
```

2.6.2. 시스템 모델

상기 데이터 모델을 활용하여 TOE가 식별 및 인증 관련 보안정책을 수행하기 위해 필요한 시스템 모델은 아래와 같다. 'RegistProcessID'는 커널에 의해서만 호출되며, 프로세스 식별자와 비밀 값을 등록하는 기능이다.

```
definition RegistProcessID :: "SecretValue ⇒ (IDSV × bool)" where
  "RegistProcessID kValue =
  (
    if kValue = KernelSecret then
      let returnID = GenerateID kValue in
      (if snd returnID = True then
        let returnSV = GenerateSecretValue kValue in
        (if snd returnSV = True then
          (
            let returnDupCheck = CheckDupID (fst returnID) kValue in
            if returnDupCheck = True then
              do{
                if storeToKernelArea kValue (fst returnID) (fst returnSV) then
                  (((fst returnID), (fst returnSV)), True)
                else
                  ((0::nat, 0::nat), False)
              }
            else
              ((0::nat, 0::nat), False)
          )
        else
          ((0::nat, 0::nat), False)
        )
      else
          ((0::nat, 0::nat), False)
      )
    else
      ((0::nat, 0::nat), False)
  )"
```

‘GenerateID’ 는 ‘RegistProcessID’ 에 의해 호출되며 프로세스 식별자를 생성하는 기능이다.

```
definition GenerateID :: "SecretValue ⇒ ID × bool" where
  "GenerateID kValue =
  (
    if kValue = KernelSecret then
      ((pseudo_random seedIDlist),True)
    else
      (seedIDlist,False)
  )"
```

‘GerenateSecretValue’ 는 ‘RegistProcessID’ 에 의해 호출되며 프로세스 비밀 값을 생성하는 기능이다.

```
definition GenerateSecretValue :: "SecretValue ⇒ SecretValue × bool" where
  "GenerateSecretValue kValue =
  (
    if kValue = KernelSecret then
      ((pseudo_random seedSVlist), True)
    else
      (seedSVlist, False)
  )"
```

‘StoreToKernelArea’ 는 ‘RegistProcessID’ 에 의해 호출되며, 앞서 생성한 프로세스 식별자와 프로세스 비밀 값을 커널 영역 저장 공간에 저장하는 기능이다.

```
definition storeToKernelArea_state :: "SecretValue ⇒ ID ⇒ SecretValue ⇒ (IDSV_list_t, bool) state"
  where "storeToKernelArea_state kValue id_in sv_in = State (λk. (
  (
    if kValue = KernelSecret then
      True
    else
      False
  )
  ,
  (
    if kValue = KernelSecret then
      ((id_in, sv_in)#IDSV_list)
    else
      IDSV_list
  )))"

definition storeToKernelArea :: "SecretValue ⇒ ID ⇒ SecretValue ⇒ bool"
  where "storeToKernelArea kValue id_in sv_in =
  (
    if fst (run_state (storeToKernelArea_state kValue id_in sv_in) IDSV_list) then
      True
    else
      False
  )"
```

‘CheckDupID’ 는 전달받은 식별 번호의 및 비밀 값의 중복을 확인하는 기능이고, 커널에 의해서만 호출이 가능하다.

```
definition CheckDupID :: "ID ⇒ SecretValue ⇒ bool" where
  "CheckDupID id_in kValue =
  (
    if kValue = KernelSecret then
      if id_in ∉ set (get_ID_list_from_IDSV_list IDSV_list []) then
        True
      else
        False
    else
      False
  )"
```

‘ReadFromKernelArea’ 는 커널 영역에 저장된 식별 번호 및 비밀 값을 불러오는 기능이다.

```
definition ReadFromKernelArea_state :: "SecretValue ⇒ nat ⇒ (IDSV, bool) state"
  where "ReadFromKernelArea_state kValue index = State (λk. (
  (if kValue = KernelSecret then
    True
  else
    False
  ),(if kValue = KernelSecret then
    (IDSV_list!index)
  else
    (0,0)
  )))"

definition ReadFromKernelArea :: "SecretValue ⇒ nat ⇒ IDSV × bool"
  where "ReadFromKernelArea kValue index =
  do {
  let tmp = (run_state (ReadFromKernelArea_state kValue index) (0,0)) in
  (if fst tmp then
    (snd tmp, True)
  else
    (snd tmp, False)
  )
  }"
```

‘CheckIDSecret’ 은 전달받은 식별 번호와 비밀 값을 확인하는 기능이다. 일치할 경우 참을, 불일치할 경우 거짓을 반환한다.

```
definition CheckIDSecret :: "SecretValue ⇒ ID ⇒ SecretValue ⇒ bool" where
  "CheckIDSecret kSecret id_in sv_in ≡
  (if kSecret = KernelSecret then
    find_IDSV_in_IDSV_list kSecret id_in sv_in IDSV_list
  else
    False
  )"
```

‘SuspiciousLog’ 는 인증 실패에 대한 로그를 기록하는 기능이다.

```
definition SuspiciousLog :: "SecretValue ⇒ ID ⇒ SecretValue ⇒ timestamp ⇒ nat ⇒ Log ⇒ Logs"
  where "SuspiciousLog kSecret id_in sv_in ts_in trials log_in ≡
  (if trials < 5 then
    (if CheckIDSecret kSecret id_in sv_in then
      (if CheckLogSize log_file then
        log_in#log_file
      else
        log_file)
    else
      log_file
    )
  else
    let tmp = BanId kSecret id_in sv_in ts_in in
    log_file
  )"
```

‘CheckLogSize’ 는 로그 파일 크기 확인하는 기능이다.

```
definition CheckLogSize :: "Logs ⇒ bool"
  where "CheckLogSize log_in =
  (if length log_in < 9999 then
    True
  else
    False
  )"
```

'BanId' 는 식별 번호 및 비밀 값을 차단하는 기능이다. 일정 횟수로 인증이 실패하였을 때, 영구적으로 인증을 방지한다.

```
definition BanId :: "SecretValue ⇒ ID ⇒ SecretValue ⇒ timestamp ⇒ BanList × bool"
  where "BanId kSecret id_in sv_in ts_in ≡
  (if kSecret = KernelSecret then
    ((id_in,sv_in)#ban_list, True)
  else
    (ban_list, False)
  )"
```

2.6.3. 검증 결과

① 최초 논리식

```
definition RegistProcessID :: "SecretValue ⇒ (IDSV × bool)" where
  "RegistProcessID kValue =
  (
    if kValue = KernelSecret then
      let returnID = GenerateID kValue in
      (if snd returnID = True then
        let returnSV = GenerateSecretValue kValue in
        (if snd returnSV = True then
          (
            let returnDupCheck = CheckDupID (fst returnID) kValue in
            if returnDupCheck = True then
              do{
                if storeToKernelArea kValue (fst returnID) (fst returnSV) then
                  (((fst returnID), (fst returnSV)), True)
                else
                  ((0::nat, 0::nat), False)
              }
            else
              ((0::nat, 0::nat), False)
          )
        else
          ((0::nat, 0::nat), False)
        )
      else
          ((0::nat, 0::nat), False)
      )
    else
      ((0::nat, 0::nat), False)
   )"

definition GenerateID :: "SecretValue ⇒ ID × bool" where
  "GenerateID kValue =
  (
    if kValue = KernelSecret then
      ((pseudo_random seedIDlist),True)
    else
      (seedIDlist,False)
  )"

definition SuspiciousLog :: "SecretValue ⇒ ID ⇒ SecretValue ⇒ timestamp ⇒ nat ⇒ Log ⇒ Logs"
  where "SuspiciousLog kSecret id_in sv_in ts_in trials log_in ≡
  (if trials < 5 then
    (if CheckIDSecret kSecret id_in sv_in then
      (if CheckLogSize log_file then
        log_in#log_file
      else
        log_file)
    else
      log_file
    )
  else
    let tmp = BanId kSecret id_in sv_in ts_in in
    log_file
  )"
```

```
definition BanId :: "SecretValue ⇒ ID ⇒ SecretValue ⇒ timestamp ⇒ BanList × bool"
  where "BanId kSecret id_in sv_in ts_in ≡
  (if kSecret = KernelSecret then
    ((id_in,sv_in)#ban_list, True)
  else
    (ban_list, False)
  )"
definition GenerateSecretValue :: "SecretValue ⇒ SecretValue × bool" where
  "GenerateSecretValue kValue =
  (
    if kValue = KernelSecret then
      ((pseudo_random seedSVlist), True)
    else
      (seedSVlist, False)
  )"
definition storeToKernelArea_state :: "SecretValue ⇒ ID ⇒ SecretValue ⇒ (IDSV_list_t, bool) state"
  where "storeToKernelArea_state kValue id_in sv_in = State (λk. (
  (
    if kValue = KernelSecret then
      True
    else
      False
  )
  ,
  (
    if kValue = KernelSecret then
      ((id_in, sv_in)#IDSV_list)
    else
      IDSV_list
  )))"

definition storeToKernelArea :: "SecretValue ⇒ ID ⇒ SecretValue ⇒ bool"
  where "storeToKernelArea kValue id_in sv_in =
  (
    if fst (run_state (storeToKernelArea_state kValue id_in sv_in) IDSV_list) then
      True
    else
      False
  )"
definition CheckDupID :: "ID ⇒ SecretValue ⇒ bool" where
  "CheckDupID id_in kValue =
  (
    if kValue = KernelSecret then
      if id_in ∉ set (get_ID_list_from_IDSV_list IDSV_list []) then
        True
      else
        False
    else
      False
  )"
```

```
definition ReadFromKernelArea_state :: "SecretValue ⇒ nat ⇒ (IDSV, bool) state"
  where "ReadFromKernelArea_state kValue index = State (λk. (
  (if kValue = KernelSecret then
    True
  else
    False
  ),(if kValue = KernelSecret then
    (IDSV_list!index)
  else
    (0,0)
  )))"

definition ReadFromKernelArea :: "SecretValue ⇒ nat ⇒ IDSV × bool"
  where "ReadFromKernelArea kValue index =
  do {
  let tmp = (run_state (ReadFromKernelArea_state kValue index) (0,0)) in
  (if fst tmp then
    (snd tmp, True)
  else
    (snd tmp, False)
  )
  }"

definition CheckIDSecret :: "SecretValue ⇒ ID ⇒ SecretValue ⇒ bool" where
  "CheckIDSecret kSecret id_in sv_in ≡
  (if kSecret = KernelSecret then
    find_IDSV_in_IDSV_list kSecret id_in sv_in IDSV_list
  else
    False
  )"

definition CheckLogSize :: "Logs ⇒ bool"
  where "CheckLogSize log_in =
  (if length log_in < 9999 then
    True
  else
    False
  )"
```

② 결과

consts RegistProcessID :: “nat ⇒ (nat × nat) × bool”
consts GenerateID :: “nat ⇒ nat × bool”
consts GenerateSecretValue :: “nat ⇒ nat × bool”
consts storeToKernelArea :: “nat ⇒ nat ⇒ nat ⇒ bool”
consts CheckDupID :: “nat ⇒ nat ⇒ bool”
consts ReadFromKernelArea :: “nat ⇒ nat ⇒ (nat × nat) × bool”
consts CheckIDSecret :: “nat ⇒ nat ⇒ nat ⇒ bool”
consts SuspiciousLog :: “nat ⇒ nat ⇒ nat ⇒ timestamp ⇒ nat ⇒ char list ⇒ char list list”
consts CheckLogSize :: “char list list ⇒ bool”
consts BanId :: “nat ⇒ nat ⇒ nat ⇒ timestamp ⇒ (nat × nat) list × bool”

2.7. 보안속성 관리 관련 보안정책모델

본 보안정책모델은 TOE가 보안속성을 관리하기 위해 필요하다. TOE는 보안정책모델에 따라 보안속성을 관리하며 점검할 수 있다.

2.7.1. 데이터 모델

본 보안정책모델에서 필요한 데이터 모델은 아래와 같다. ‘f_Name’의 타입은 유형을 가지고, 보안을 위해 감사해야할 함수의 집합체로 구성되어 있다. ‘log_file’은 log를 남길 데이터의 저장소로 string의 list로 정의한다. 마지막으로, ‘sFunctions’는 ‘fName’의 set으로 정의되며, 현재 프로세스가 설정된다.

```
datatype fName =
ch_threads_list | ch_threads_queue | ch_thread | ch_virtual_timer
ch_virtual_timers_list | ch_ready_list | ch_system_debug |
ch_system | firstprio | currp | list_init | list_isempty |
list_notempty | queue_init | queue_isempty | queue_notempty |
list_insert | list_remove | queue_prio_insert | queue_fifo_remove
queue_lifo_remove | queue_dequeue | chSchIsRescRequiredI |
chSchCanYieldS | chSchPreemption

definition log_file :: "string list"
  where "log_file = ['''']"

definition sFunctions :: "fName set"
  where "sFunctions = {currp}"
```

2.7.2. 시스템 모델

상기 데이터 모델을 활용하여 TOE가 보안속성 관리 관련 보안정책을 수행하기 위해 필요한 시스템 모델은 아래와 같다. ‘SecurityFunctionCheck’는 확인하고자 하는 하나의 함수 이름을 ‘f_Name’ 타입으로 입력받고, 입력받은 함수의 이름이 ‘s_Functions’에 존재하는지 점검하는 모델이다.

```
definition SecurityFunctionCheck :: "fName ⇒ bool"
  where "SecurityFunctionCheck fname_in =
  (if fname_in ∈ sFunctions then
    True
  else
    False)"
```

‘FunctionAuditing’은 관리하고자 하는 함수 이름을 ‘fName’ 타입으로 입력받고, 함수의 행동인 ‘action’을 입력받아 ‘log_file’에 기록하여 커널의 보안 동작을 점검한다.

```
definition FunctionAuditing :: "fName ⇒ string ⇒ string list"
  where "FunctionAuditing fname_in action= action#log_file"
```

‘DataAudit’은 커널의 보안 속성 파일을 제어하는 기능이다.

```
definition DataAuditing :: "(unit, unit) state"
  where "DataAuditing = return ()"
```

‘TestSuit’는 미리 정의된 Test Case로 보안성을 확인하는 기능이다.

```
definition TestSuit :: "address ⇒ bool"
  where "TestSuit addr_in = True"
```

‘PermissionCheck’는 특정 메모리 영역에 접근, 읽기, 쓰기를 수행할 때 접근하는 프로세스가 해당 메모리 영역에 대한 권한을 만족하는지 확인하는 기능이다.

```
definition PermissionCheck :: "address ⇒ bool"
  where "PermissionCheck addr_in =
  (if permission_list!addr_in = Kernel then
    True
  else
    False
  )"
```

2.7.3. 검증결과

① 최초 논리식

```
datatype fName =
ch_threads_list | ch_threads_queue | ch_thread | ch_virtual_timer
ch_virtual_timers_list | ch_ready_list | ch_system_debug |
ch_system | firstprio | currp | list_init | list_isempty |
list_notempty | queue_init | queue_isempty | queue_notempty |
list_insert | list_remove | queue_prio_insert | queue_fifo_remove
queue_lifo_remove | queue_dequeue | chSchIsRescRequiredI |
chSchCanYieldS | chSchPreemption

definition log_file :: "string list"
  where "log_file = ['''']"


definition sFunctions :: "fName set"
  where "sFunctions = {currp}"

definition SecurityFunctionCheck :: "fName ⇒ bool"
  where "SecurityFunctionCheck fname_in =
  (if fname_in ∈ sFunctions then
    True
  else
    False)"
definition FunctionAuditing :: "fName ⇒ string ⇒ string list"
  where "FunctionAuditing fname_in action= action#log_file"

definition DataAuditing :: "(unit, unit) state"
  where "DataAuditing = return ()"

definition TestSuit :: "address ⇒ bool"
  where "TestSuit addr_in = True"

definition PermissionCheck :: "address ⇒ bool"
  where "PermissionCheck addr_in =
  (if permission_list!addr_in = Kernel then
    True
  else
    False
  )"
```

② 결과

consts SecurityFunctionCheck :: “fName ⇒ bool”

consts sFunctions :: “fName set”

consts SecurityFunctionCheck :: “fName ⇒ bool”

consts FunctionAuditing :: “fName ⇒ char list ⇒ char list list”

consts DataAuditing :: “(unit, unit) state“

consts TestSuit :: “nat ⇒ bool “

consts PermissionCheck :: “nat ⇒ bool“

3. 보안정책모델과 보안기능 요구사항 간의 추적성

본 장에서는 앞선 2장에서 6장까지 설명한 각 보안정책모델들과 ST의 보안기능 요구사항 간 추적성에 대하여 설명한다. 모든 보안기능 요구사항들은 TOE의 보안정책모델에 대해서 아래 [표 1]과 같은 추적성을 갖는다.

보안기능 요구사항	보안정책모델					
	감사	자체보호	자원활용	데이터 보호	식별 및 인증	기능 관리
FAU_GEN.1	X					
FAU_STG.1	X					
FAU_STG.3	X					
FAU_STG.4	X					
FDP_IFC.1				X		
FDP_IFF.1				X		
FDP_RIP.1				X		
FIA_UAU.1					X	
FIA_UID.1					X	
FMT_MOF.1						X
FMT_MSA.1						X
FMT_MSA.3						X
FMT_MTD.1						X
FMT_SMF.1						X
FMT_SMR.1						X
FPT_FLS.1		X				
FPT_RCV.1		X				
FPT_TST.1		X				
FRU_PRS.1			X			
FRU_RSA.1			X			

[표 1] 보안목적과 보안기능요구사항의 추적성