SecCSL:
Security Concurrent Separation Logic

Gidon Ernst [1] Toby Murray [2]

[1]LMU Munich, Germany gidon.ernst@lmu.de

[2]University of Melbourne, Australia toby.murray@unimelb.edu.au

Elite-SE Ringvorlesung, 4. July 2019

“40M credit cards hacked” [CNNMoney 2005]

Kreditkartendaten ungeschützt im Netz
(öffentlicher Zugriff auf private Daten)

“2 Billion Records Exposed In Massive Smart Home Device Breach” [Forbes, 2. Juli 2019]

Nutzerdaten ungeschützt im Netz
(kein Passwort)

Facebook: Shadow Profile Leak [2015]

Nutzerprofile ungeschützt im Netz
(private Daten falsch zugeordnet)

Remote timing attacks are practical
[Brumley & Boneh, Usenix Security 2003]

Rekonstruktion privater Schlüssel
durch Messen von Antwortzeiten des Servers

✗ Information Leak:
Sensitive Quelle $\rightsquigarrow$ Öffentliche Senke

```
bool check(char *input, char *password) {
    ...
    log("incorrect password: %s\n", input);
    ...
}
```

✗ Timing Leak:
Laufzeit hängt von sensitiven Daten ab

```
bool check(char *input, char *password) {
    return strcmp(input, password) == 0;
}
```

✗ Timing Leak:
Laufzeit hängt von sensitiven Daten ab

```
bool check(char *input, char *password) {
    for(int i=0; i<strlen(password); i++) {
        if(input[i] != password[i])
            return false;
    }
    return true;
}
```

✗ Timing Leak:
Laufzeit hängt von sensitiven Daten ab

```
bool check(char *input, char *password) {
    for(int i=0; i<strlen(password); i++) {
        if(input[i] != password[i])
            return false;
    }
    return true;
}
```

- Schleife läuft $n\times \rightarrow$ erste n Zeichen korrekt
- Angriff linear in $O(\texttt{sizeof(char)} \cdot \texttt{strlen(password)})$

✗ Timing Leak:
Laufzeit hängt von sensitiven Daten ab

```
bool check(char *input, char *password) {
    for(int i=0; i<strlen(password); i++) {
        if(input[i] != password[i])
            return false;
    }
    return true;
}
```

- Schleife läuft $n\times \rightarrow$ erste n Zeichen korrekt
- Angriff linear in $O($`sizeof(char)` $\cdot$ `strlen(password)`$)$
- Bug falls `strlen(input) != strlen(password)`

seL4: Information Flow Enforcement [Murray et al, S&P 2013]

Beweis

- ✓ Vollständige Isolation zwischen Prozessen
- ✓ Basis: funktionale Korrektheit [Klein et al, SOSP 2009]

seL4: Information Flow Enforcement [Murray et al, S&P 2013]

Beweis

- ✓ Vollständige Isolation zwischen Prozessen
- ✓ Basis: funktionale Korrektheit [Klein et al, SOSP 2009]
- ✗ Automatisierung (Pointer, Concurrency)
- ✗ Kein spezielles Tool (Isabelle)

Continuous formal verification of Amazon s2n [Chudnov et al, CAV 2018]

TLS Bibliothek

- ✓ Formale Spezifikation mit Cryptol [Lewis, FMSE 2007]
- ✓ Korrektheitsbeweis mit SAW [Galois, SIGAda 2013]
- ✓ CI: Re-Verifikation bei Änderungen

Continuous formal verification of Amazon s2n [Chudnov et al, CAV 2018]

TLS Bibliothek

- ✓ Formale Spezifikation mit Cryptol [Lewis, FMSE 2007]
- ✓ Korrektheitsbeweis mit SAW [Galois, SIGAda 2013]
- ✓ CI: Re-Verifikation bei Änderungen (aktuell: build error)

Continuous formal verification of Amazon s2n [Chudnov et al, CAV 2018]

TLS Bibliothek

- ✓ Formale Spezifikation mit Cryptol [Lewis, FMSE 2007]
- ✓ Korrektheitsbeweis mit SAW [Galois, SIGAda 2013]
- ✓ CI: Re-Verifikation bei Änderungen (aktuell: build error)
- ? Timing Angriffe? [Albrecht & Paterson, Eurocrypt 2016]
- ✗ Bounded Non-Modular Verification

Ziel: Security-Beweise für Systemcode in C

Challenges

- Integration Korrektheit + Security
- Ausdrucksstarke Security Eigenschaften
- Low-level Memory & Concurrency
- Automatisierung von Beweisen

Ziel: Security-Beweise für Systemcode in C

Challenges

- Integration Korrektheit + Security
- Ausdrucksstarke Security Eigenschaften
- Low-level Memory & Concurrency
- Automatisierung von Beweisen

✓ SecCSL: Security Concurrent Separation Logic
[Ernst & Murray, CAV 2019]

Outline

- ✓ Motivation: Verifikation von Information Security
- ▶ Security = Noninterference
- ☐ Beispiel
- ☐ SecC: Tool Demo
- ☐ SecCSL: Security Concurrent Separation Logic

Noninterference [Goguen & Meseguer, S&P 1982]

Klassifikation von Daten: `low` (öffentlich), `high` (geheim)

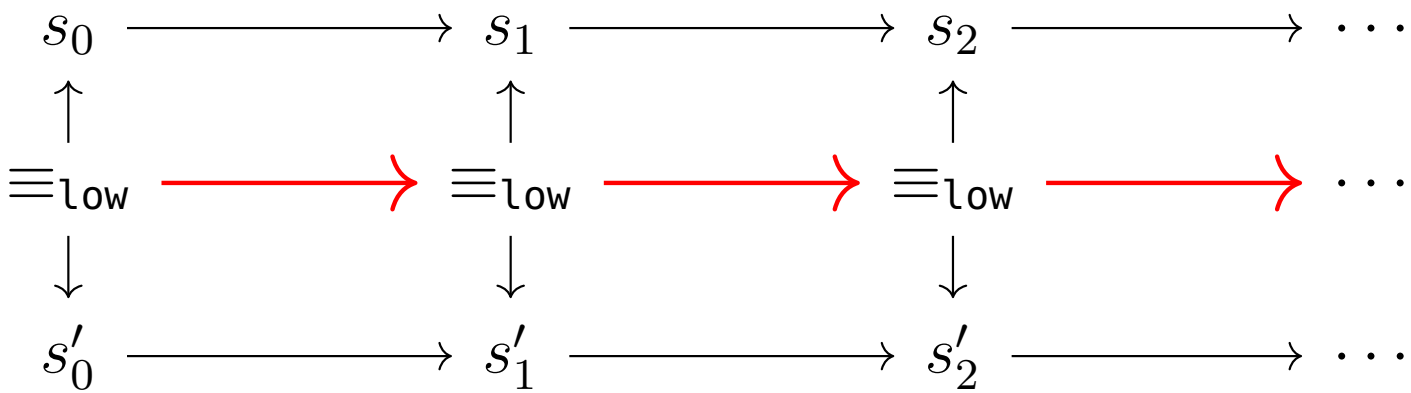
Semantisches Kriterium *Noninterference*: $\texttt{high} \not\rightsquigarrow \texttt{low}$

Noninterference [Goguen & Meseguer, S&P 1982]

Klassifikation von Daten: `low` (öffentlich), `high` (geheim)

Semantisches Kriterium *Noninterference*: `high` $\not\rightsquigarrow$ `low`

Vergleich von zwei Ausführungen mit `low`-*Bisimulation*

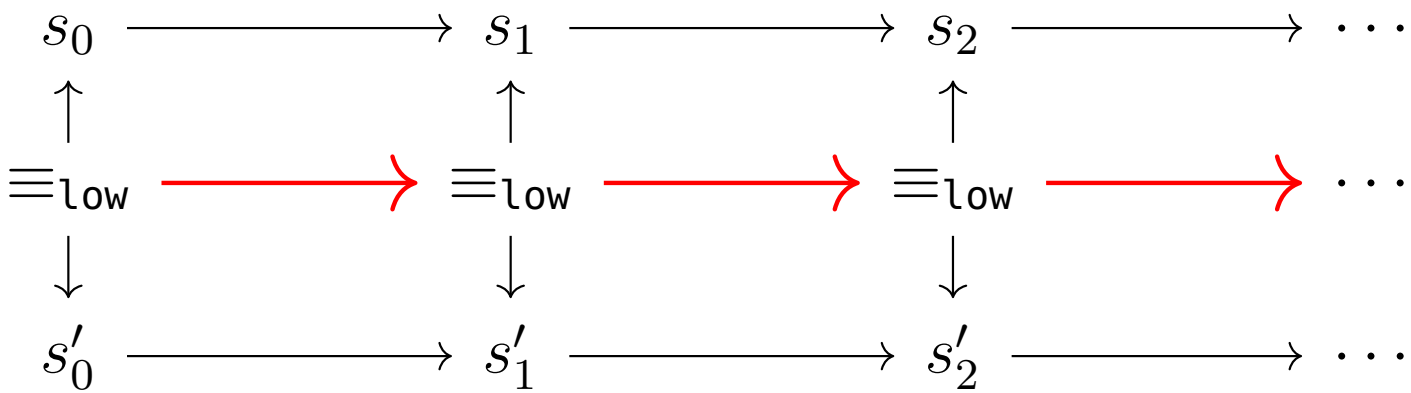


Noninterference [Goguen & Meseguer, S&P 1982]

Klassifikation von Daten: `low` (öffentlich), `high` (geheim)

Semantisches Kriterium *Noninterference*: $\texttt{high} \not\rightsquigarrow \texttt{low}$

Vergleich von zwei Ausführungen mit `low`-*Bisimulation*



$\rightarrow$ Öffentliche Daten hängen nicht von geheimen Daten ab

Noninterference: Beispiel

$$\{x :: \texttt{low} \land y :: \texttt{high}\} \tag{1}$$

(2)

Noninterference: Beispiel

$$\{x :: \texttt{low} \land y :: \texttt{high}\} \tag{1}$$

$$\tag{2}$$

Relationale Semantik über Zustandspaare

- $[\![x :: \texttt{low}]\!] \equiv (x = x')$ und $[\![x :: \texttt{high}]\!] \equiv \texttt{true}$

Noninterference: Beispiel

$\{x :: \texttt{low} \wedge y :: \texttt{high}\}$

```
z = x + 1;
```

(1)

(2)

Relationale Semantik über Zustandspaare

- $[\![x :: \texttt{low}]\!] \equiv (x = x')$ und $[\![x :: \texttt{high}]\!] \equiv \texttt{true}$

Noninterference: Beispiel

$\{x :: \texttt{low} \land y :: \texttt{high}\}$

```
z = x + 1;
```

(1)

(2)

Relationale Semantik über Zustandspaare

- ▶ $[\![x :: \texttt{low}]\!] \equiv (x = x')$ und $[\![x :: \texttt{high}]\!] \equiv \texttt{true}$
- ✓ $x = x' \implies x + 1 = x' + 1$

Noninterference: Beispiel

$$\{x :: \texttt{low} \land y :: \texttt{high}\} \tag{1}$$

```
z = x + 1;
```

$$\{x :: \texttt{low} \land y :: \texttt{high} \land z :: \texttt{low}\} \tag{2}$$

Relationale Semantik über Zustandspaare

- ▶ $[\![x :: \texttt{low}]\!] \equiv (x = x')$ und $[\![x :: \texttt{high}]\!] \equiv \texttt{true}$
- ✓ $x = x' \implies x + 1 = x' + 1$

Noninterference: Beispiel

$\{x :: \texttt{low} \wedge y :: \texttt{high}\}$

```
z = x + 1;
```
(1)

$\{x :: \texttt{low} \wedge y :: \texttt{high} \wedge z :: \texttt{low}\}$

```
z = z + y;
```
(2)

Relationale Semantik über Zustandspaare

- ▶ $[\![x :: \texttt{low}]\!] \equiv (x = x')$ und $[\![x :: \texttt{high}]\!] \equiv \texttt{true}$
- ✓ $x = x' \implies x + 1 = x' + 1$

Noninterference: Beispiel

$\{x :: \texttt{low} \wedge y :: \texttt{high}\}$

```
z = x + 1;
```
(1)

$\{x :: \texttt{low} \wedge y :: \texttt{high} \wedge z :: \texttt{low}\}$

```
z = z + y;
```
(2)

Relationale Semantik über Zustandspaare

- $[\![x :: \texttt{low}]\!] \equiv (x = x')$ und $[\![x :: \texttt{high}]\!] \equiv \texttt{true}$
- ✓ $x = x' \implies x + 1 = x' + 1$
- ✗ $z = z' \not\implies z + y = z' + y'$

Noninterference: Beispiel

$\{x :: \texttt{low} \wedge y :: \texttt{high}\}$

```
z = x + 1;
```
(1)

$\{x :: \texttt{low} \wedge y :: \texttt{high} \wedge z :: \texttt{low}\}$

```
z = z + y;
```
(2)

$\{x :: \texttt{low} \wedge y :: \texttt{high} \wedge z :: \texttt{high}\}$

Relationale Semantik über Zustandspaare

- $[\![x :: \texttt{low}]\!] \equiv (x = x')$ und $[\![x :: \texttt{high}]\!] \equiv \texttt{true}$
- ✓ $x = x' \implies x + 1 = x' + 1$
- ✗ $z = z' \not\implies z + y = z' + y'$

Outline

- ✓ Motivation: Verifikation von Information Security
- ✓ Noninterference
- ▶ Beispiel
- ☐ SecC: Tool Demo
- ☐ SecCSL: Security Concurrent Separation Logic

Beispiel: Concurrent Information Flow

```
/* globals shared between the two threads */
struct record { bool is_classified; int data; };
struct record * rec = ...;

/* memory-mapped IO device register */;
volatile int * const OUTPUT_REG = ...;
```

Beispiel: Concurrent Information Flow

```
/* globals shared between the two threads */
struct record { bool is_classified; int data; };
struct record * rec = ...;

/* memory-mapped IO device register */;
volatile int * const OUTPUT_REG = ...;


/* thread 1: output the record */

while(true) {
    lock(mutex);
    if (!rec->is_classified)
        *OUTPUT_REG = rec->data;
    unlock(mutex);
}
```

Beispiel: Concurrent Information Flow

```
/* globals shared between the two threads */
struct record { bool is_classified; int data; };
struct record * rec = ...;

/* memory-mapped IO device register */;
volatile int * const OUTPUT_REG = ...;
```

```
/* thread 1: output the record */

while(true) {
    lock(mutex);
    if (!rec->is_classified)
        *OUTPUT_REG = rec->data;
    unlock(mutex);
}
```

```
/* thread 2: edit the record */

lock(mutex);
/* remove classification */
rec->is_classified = FALSE;
/* erase previous data */
rec->data = 0;
unlock(mutex);
```

Beispiel: Concurrent Information Flow

```
/* globals shared between the two threads */
struct record { bool is_classified; int data; };
struct record * rec = ...;

/* memory-mapped IO device register */;
volatile int * const OUTPUT_REG = ...;
```

```
/* thread 1: output the record */

while(true) {
    lock(mutex);
    if (!rec->is_classified)
        *OUTPUT_REG = rec->data;
    unlock(mutex);
}
```

```
/* thread 2: edit the record */

lock(mutex);
/* remove classification */
rec->is_classified = FALSE;
/* erase previous data */
rec->data = 0;
unlock(mutex);
```

Invarianten?

Beispiel: Concurrent Information Flow

```
/* globals shared between the two threads */
struct record { bool is_classified; int data; };
struct record * rec = ...;

/* memory-mapped IO device register */;
volatile int * const OUTPUT_REG = ...;
```

```
/* thread 1: output the record */

while(true) {
    lock(mutex);
    if (!rec->is_classified)
        *OUTPUT_REG = rec->data;
    unlock(mutex);
}
```

```
/* thread 2: edit the record */

lock(mutex);
/* remove classification */
rec->is_classified = FALSE;
/* erase previous data */
rec->data = 0;
unlock(mutex);
```

Invarianten?

- Gültige Pointer: `rec`, `OUTPUT_REG`
- Wechselseitiger Ausschluss
- `rec->is_classified == 0` $\Longrightarrow$ `rec->data` $::$ `low`

Beispiel: Concurrent Information Flow

```
/* globals shared between the two threads */
struct record { bool is_classified; int data; };
struct record * rec = ...;

/* memory-mapped IO device register */;
volatile int * const OUTPUT_REG = ...;
```

```
/* thread 1: output the record */

while(true) {
    lock(mutex);
    if (!rec->is_classified)
        *OUTPUT_REG = rec->data;
    unlock(mutex);
}
```

```
/* thread 2: edit the record */

lock(mutex);
/* remove classification */
rec->is_classified = FALSE;
/* erase previous data */
rec->data = 0;
unlock(mutex);
```

Invarianten in SecCSL

$$\exists\, c, d.\ \texttt{rec} \mapsto (c, d) \wedge c :: \texttt{low} \wedge (\neg\, c \Rightarrow d :: \texttt{low}) \tag{3}$$

$$\exists\, v.\ \texttt{OUTPUT\_REG} \overset{\texttt{low}}{\longmapsto} v \wedge v :: \texttt{low} \tag{4}$$

Outline

- ✓ Motivation: Verifikation von Information Security
- ✓ Noninterference
- ✓ Beispiel
- ▶ SecC: Tool Demo
- ☐ SecCSL: Security Concurrent Separation Logic

Outline

- ✓ Motivation: Verifikation von Information Security
- ✓ Noninterference
- ✓ Beispiel
- ✓ SecC: Tool Demo
- ▶ SecCSL: Security Concurrent Separation Logic

Separation Logic (SL) [Reynolds, LICS 2002]

- Erweiterung von Hoare Logic $\{P\}\ c\ \{Q\}$

Separation Logic (SL) [Reynolds, LICS 2002]

- Erweiterung von Hoare Logic $\{P\}\ c\ \{Q\}$
- Points-to $\texttt{x} \mapsto a$ für Speicherzelle `int` $*$x:
  - Impliziert x $\neq$ NULL und $*\texttt{x} = a$
  - *Ownership*: Kein anderer Thread greift auf x zu

Separation Logic (SL) [Reynolds, LICS 2002]

- Erweiterung von Hoare Logic $\{P\}\ c\ \{Q\}$
- Points-to $\mathsf{x} \mapsto a$ für Speicherzelle `int` $*$x:
  - Impliziert x $\neq$ NULL und $*\mathsf{x} = a$
  - *Ownership*: Kein anderer Thread greift auf x zu
- Separating conjunction $P \star Q$
  - P und Q beschreiben disjunkte Speicherbereiche
  - *Separation*: Lokaler Effekt von Speicherzugriffen
    $\{\mathsf{x} \mapsto a \star Q\}\ *\mathsf{x} = b\ \{\mathsf{x} \mapsto b \star Q\}$

Separation Logic (SL) [Reynolds, LICS 2002]

- Erweiterung von Hoare Logic $\{P\}\ c\ \{Q\}$
- Points-to $\mathsf{x} \mapsto a$ für Speicherzelle `int` $*$x:
  - Impliziert x $\neq$ NULL und $*\mathsf{x} = a$
  - *Ownership*: Kein anderer Thread greift auf x zu
- Separating conjunction $P \star Q$
  - P und Q beschreiben disjunkte Speicherbereiche
  - *Separation*: Lokaler Effekt von Speicherzugriffen
    $\{\mathsf{x} \mapsto a \star Q\}\ *\mathsf{x} = b\ \{\mathsf{x} \mapsto b \star Q\}$
- Verkettete Datenstrukturen

Concurrent SL [O'Hearn, CONCUR 2004]

- Parallele Komposition

$$\frac{\{P_1\}\ c_1\ \{Q_1\} \quad \{P_2\}\ c_2\ \{Q_2\}}{\{P_1 \star P_2\}\ c_1 \parallel c_2\ \{Q_1 \star Q_2\}}$$

Concurrent SL [O'Hearn, CONCUR 2004]

- Parallele Komposition

$$\frac{\{P_1\}\ c_1\ \{Q_1\} \quad \{P_2\}\ c_2\ \{Q_2\}}{\{P_1 \star P_2\}\ c_1 \parallel c_2\ \{Q_1 \star Q_2\}}$$

- Locking = Ownership-Transfer
  Resourceninvariante $\mathrm{inv}(l)$ für Lock l

$$\frac{}{\{P\}\ \texttt{lock}(l)\ \{\mathrm{inv}(l) \star P\}} \qquad \frac{}{\{\mathrm{inv}(l) \star P\}\ \texttt{unlock}(l)\ \{P\}}$$

Concurrent SL [O'Hearn, CONCUR 2004]

- Parallele Komposition

$$\frac{\{P_1\}\ c_1\ \{Q_1\} \quad \{P_2\}\ c_2\ \{Q_2\}}{\{P_1 \star P_2\}\ c_1 \parallel c_2\ \{Q_1 \star Q_2\}}$$

- Locking = Ownership-Transfer
  Resourceninvariante $\mathrm{inv}(l)$ für Lock l

$$\frac{}{\{P\}\ \texttt{lock}(l)\ \{\mathrm{inv}(l) \star P\}} \qquad \frac{}{\{\mathrm{inv}(l) \star P\}\ \texttt{unlock}(l)\ \{P\}}$$

- Lock-basierte Concurrency

Security Concurrent Separation Logic (SecCSL) [Ernst & Murray, CAV 2019]

- Security-Labels $\ell \in \{\texttt{low}, \ldots, \texttt{high}\}$

Security Concurrent Separation Logic (SecCSL) [Ernst & Murray, CAV 2019]

- Security-Labels $\ell \in \{\texttt{low}, \ldots, \texttt{high}\}$
- Informations-Quellen:
  - Klassifikation von *Werten* $v :: \ell$
  - $v :: \texttt{low}$: Daten v sind öffentlich
  - $v :: \texttt{high} \iff \texttt{true}$
  - $v :: (c \,?\, \texttt{high} : \texttt{low})$ datenabhägig

Security Concurrent Separation Logic (SecCSL) [Ernst & Murray, CAV 2019]

- Security-Labels $\ell \in \{\texttt{low}, \ldots, \texttt{high}\}$
- Informations-Quellen:
  - Klassifikation von *Werten* $v :: \ell$
  - $v :: \texttt{low}$: Daten v sind öffentlich
  - $v :: \texttt{high} \iff \texttt{true}$
  - $v :: (c \,?\, \texttt{high} : \texttt{low})$ datenabhägig
- Informations-Senken:
  - Klassifikation von *Speicherzellen* $\mathsf{x} \overset{\ell}{\mapsto} a$
  - $\mathsf{x} \overset{\texttt{low}}{\longmapsto} a$: $*\mathsf{x} = b$ erfordert $b :: \texttt{low}$
  - $\mathsf{x} \overset{\texttt{high}}{\longmapsto} a \iff \mathsf{x} \mapsto a$

Security Concurrent Separation Logic (SecCSL) [Ernst & Murray, CAV 2019]

- Security-Labels $\ell \in \{\texttt{low}, \ldots, \texttt{high}\}$
- Informations-Quellen:
  - Klassifikation von *Werten* $v :: \ell$
  - $v :: \texttt{low}$: Daten v sind öffentlich
  - $v :: \texttt{high} \iff \texttt{true}$
  - $v :: (c \,?\, \texttt{high} : \texttt{low})$ datenabhägig
- Informations-Senken:
  - Klassifikation von *Speicherzellen* $\mathsf{x} \overset{\ell}{\mapsto} a$
  - $\mathsf{x} \overset{\texttt{low}}{\longmapsto} a$: $*\mathsf{x} = b$ erfordert $b :: \texttt{low}$
  - $\mathsf{x} \overset{\texttt{high}}{\longmapsto} a \iff \mathsf{x} \mapsto a$
- Datenabhängige Security-Klassifikation

SecCSL = Sec + CSL

$$\frac{}{\{\mathsf{y} \mapsto a\}\ \mathsf{x = *y}\ \{\mathsf{x} = a \wedge \mathsf{y} \mapsto a\}}$$

$$\frac{}{\{\quad \mathsf{x} \mapsto a\}\ \mathsf{*x = } b\ \{\mathsf{x} \mapsto b\}}$$

$$\frac{\{b \wedge P\}\ c_1\ \{Q\} \quad \{\neg b \wedge P\}\ c_2\ \{Q\}}{\{\quad P\}\ \mathtt{if(}b\mathtt{)}\ c_1\ \mathtt{else}\ c_2\ \{Q\}}$$

$$\frac{\{\quad b \wedge P\}\ c\ \{\quad P\}}{\{\quad P\}\ \mathtt{while(}b\mathtt{)}\ c\ \{\neg b \wedge P\}}$$

SecCSL = Sec + CSL

$$\frac{}{\{\mathsf{y} \overset{\ell}{\mapsto} a\}\ \mathsf{x} = *\mathsf{y}\ \{\mathsf{x} = a \wedge \mathsf{y} \overset{\ell}{\mapsto} a\}}$$

$$\frac{}{\{b :: \ell \wedge \mathsf{x} \overset{\ell}{\mapsto} a\}\ *\mathsf{x} = b\ \{\mathsf{x} \overset{\ell}{\mapsto} b\}}$$

$$\frac{\{b \wedge P\}\ c_1\ \{Q\} \quad \{\neg b \wedge P\}\ c_2\ \{Q\}}{\{b :: \mathtt{low} \wedge P\}\ \mathtt{if}(b)\ c_1\ \mathtt{else}\ c_2\ \{Q\}}$$

$$\frac{\{b :: \mathtt{low} \wedge b \wedge P\}\ c\ \{b :: \mathtt{low} \wedge P\}}{\{b :: \mathtt{low} \wedge P\}\ \mathtt{while}(b)\ c\ \{\neg b \wedge P\}}$$

- *Orthogonale* Erweiterung bestehender Beweisregeln

Beweis: Thread 2

```
lock(mutex);


rec->is_classified = false;
rec->data = 0;


unlock(mutex);
```

Beweis: Thread 2

```
lock(mutex);
```

$\{\ \texttt{rec} \mapsto (c, d) \wedge c :: \texttt{low} \wedge (\neg\ c \Rightarrow d :: \texttt{low})$

$\quad \star\ \texttt{OUTPUT\_REG} \overset{\texttt{low}}{\longmapsto} v \wedge v :: \texttt{low}\ \}$

```
rec->is_classified = false;
rec->data = 0;
```

```
unlock(mutex);
```

Beweis: Thread 2

```
lock(mutex);
```

$\{\ \texttt{rec} \mapsto (c, d) \wedge c :: \texttt{low} \wedge (\neg\ c \Rightarrow d :: \texttt{low})$

$\quad \star\ \texttt{OUTPUT\_REG} \overset{\texttt{low}}{\longmapsto} v \wedge v :: \texttt{low}\ \}$

```
rec->is_classified = false;
rec->data = 0;
```

$\{\ \texttt{rec} \mapsto (\texttt{false}, 0) \wedge \texttt{false} :: \texttt{low} \wedge (\texttt{true} \Rightarrow 0 :: \texttt{low})$

$\quad \star\ \texttt{OUTPUT\_REG} \overset{\texttt{low}}{\longmapsto} v \wedge v :: \texttt{low}\ \}$

```
unlock(mutex);
```

Beweis: Thread 2

```
lock(mutex);
```
$\{\ \texttt{rec} \mapsto (c, d) \wedge c :: \texttt{low} \wedge (\neg\ c \Rightarrow d :: \texttt{low})$

$\quad \star\ \texttt{OUTPUT\_REG} \overset{\texttt{low}}{\longmapsto} v \wedge v :: \texttt{low}\ \}$

```
rec->is_classified = false;
rec->data = 0;
```
$\{\ \texttt{rec} \mapsto (\texttt{false}, 0) \wedge \texttt{false} :: \texttt{low} \wedge (\texttt{true} \Rightarrow 0 :: \texttt{low})$

$\quad \star\ \texttt{OUTPUT\_REG} \overset{\texttt{low}}{\longmapsto} v \wedge v :: \texttt{low}\ \}$

```
unlock(mutex);
```
$\square$

Zusammenfassung

- Noninterference: Security
- CSL: verkettete Datenstrukturen + Concurrency
- SecCSL:
  - Elegante Integration der beiden Theorien
  - Toolunterstützung: SecC
- Vielen Dank!

SecCSL: Challenges

- $x :: \texttt{low} \Longrightarrow f(x) :: \texttt{low}$
- Fallunterscheidungen im Beweis

$$\frac{P \Rightarrow \varphi :: \texttt{low} \quad \{\varphi \wedge P\}\ c\ \{Q\} \quad \{\neg\varphi \wedge P\}\ c\ \{Q\}}{\{P\}\ c\ \{Q\}}$$