Effective Auditing

Wilmer Ricciotti
James Cheney

University of Edinburgh

International Colloquium on Theoretical Aspects of Computing
16-19 October 2018, Stellenbosch, South Africa
**Audited computation** refers to the ability to:

- Faithfully record a description (log, trace, trail) of the computation history
- Programmatically inspect that history at any time, and in particular during the computation itself
- Take decisions based on the inspection

- Typical approaches require the programmer to instrument the code in an *ad hoc* way
- We seek to develop languages with a first-class notion of computation history
Break the glass policies (e.g. in electronic medical record systems)

Retrospective security

Access control based on the function calls in the history of the current computation

Stack inspection

Annotation of data with information disclosing its origin or the way it was computed

Provenance

Motivation
Motivation

- Break-the-glass policies (e.g. in electronic medical record systems)
  - Retrospective security
Motivation

- Break-the-glass policies (e.g. in electronic medical record systems)
  - Retrospective security
- Access-control based on the function calls in the history of the current computation
  - Stack inspection
Motivation

- Break-the-glass policies (e.g. in electronic medical record systems)
  - Retrospective security
- Access-control based on the function calls in the history of the current computation
  - Stack inspection
- Annotation of data with information disclosing its origin or the way it was computed
  - Provenance
Justification Logic

- A refinement of modal logic allowing one to express
  - What is true
  - What is known to be true and the reason why it is known to be true
    \[ \Gamma \vdash [s]A \]
- Originally defined as an axiomatic proof system à la Hilbert
- Typed lambda calculi based CH-isomorphic to JL have been introduced
A lambda calculus with primitive notions of computation history and auditing (Bavera and Bonelli, 2015)

Audited units $!_q M$ are «boxes» logging the computation history of $M$ as a trail $q$)

Example:

context
...
if 1 = 2
then M
else N

context
...
in context:
(1=2) → false

if false
then M
else N
Trails

- Trails = syntactic representation of reduction chains
Trails = syntactic representation of reduction chains

\(\text{app}(\Rightarrow, r) \Rightarrow \lambda b. 2, 6\)
Trails = syntactic representation of reduction chains

\[ ! ((\lambda a, b. \langle a, b \rangle)) \ 2 \ 6 \]
\[ \rightsquigarrow ! \text{app}(\beta, r) (\lambda b. \langle 2, b \rangle) \ 6 \]
Trails = syntactic representation of reduction chains

\[ \lambda a, b. \langle a, b \rangle \mapsto (\lambda a, b. \langle a, b \rangle) \mapsto (\lambda b. \langle 2, b \rangle) \mapsto (\lambda a, b. \langle a, b \rangle) \mapsto (\lambda b. \langle 2, b \rangle) \mapsto \]
Trails = syntactic representation of reduction chains

\[
\begin{align*}
& \downarrow ((\lambda a, b. \langle a, b \rangle)) 2 6 \\
& \sim \downarrow \text{app}(\beta, r) (\lambda b. \langle 2, b \rangle)) 6 \\
& \sim \downarrow \text{t(app}(\beta, r), \beta) \langle 2, 6 \rangle \\
\end{align*}
\]

\[
\begin{align*}
\beta : (\lambda a, b. \langle a, b \rangle) 2 & \triangleright \lambda b. \langle 2, b \rangle \\
\text{app}(\beta, r) : ((\lambda a, b. \langle a, b \rangle) 2) 6 & \triangleright (\lambda b. \langle 2, b \rangle) 6 \\
\beta : (\lambda b. \langle 2, b \rangle) 6 & \triangleright \langle 2, 6 \rangle \\
\text{t(app}(\beta, r), \beta) : ((\lambda a, b. \langle a, b \rangle) 2) 6 & \triangleright \langle 2, 6 \rangle \\
\end{align*}
\]

- Reflexivity (r), transitivity (t)
- Reduction steps (\beta)
- Congruence rules (e.g. app)
Principle:

\[ !_q \mathcal{F}[M] \rightarrow !_t(q,Q[q']) \mathcal{F}[N] \]

Where:
- \( q' : M \rightarrow N \)
- \( Q \) is a trail context corresponding to \( \mathcal{F} \)

Concrete definition: trail permutations
Trail normalization
Trail normalization

- A computation step without trails:

\[(\lambda x. M) \ N \to M^{[N/x]}\]
Trail normalization

- A computation step without trails:

\[(\lambda x. M) N \rightarrow M[^N/x]\]

- A computation step with trails:

\[(\lambda x. M) N \rightarrow \beta \triangleright M[^N/x]\]
Trail normalization

- A computation step without trails:
  $$(\lambda x. M) \ N \rightarrow M[^N/x]$$

- A computation step with trails:
  $$(\lambda x. M) \ N \rightarrow \beta \triangleright M[^N/x]$$

- The operation pushing $q'$ to the outside is called **trail normalization**
  - Defined as a series of permutation reductions
    $!_q F[M] \rightarrow !_q F[q' \triangleright N] \rightarrow !_t(q, q'[q']) F[N]$
  - Its cost depends on the size of $F$
  - An efficiency issue similar to the one related to substitution
Auditing is performed by an *inspection* operator

\[ !_q F[\iota(q)] \rightarrow !_q F[ti \triangleright q\vartheta] \]

- Trail inspection *reifies* the computation history of the *audited unit* currently being executed
- It allows us to analyse the history by primitive recursion
- Example: \(\vartheta(\beta) = \vartheta(ti) = 1\), \(\vartheta(t) = \vartheta(app) = \lambda a, b. a + b\), \(\vartheta(r) = 0\)

\[
\begin{align*}
! (t_0 & \leftarrow \iota(q)); \\
_ & \leftarrow ((\lambda a, b. (a, b)) 2) 6; \\
t_1 & \leftarrow \iota(q); \\
t_1 & - t_0 
\end{align*}
\]
Trail inspection

- Auditing is performed by an *inspection* operator

\[ !q \mathcal{F}[\iota(\vartheta)] \rightarrow !q \mathcal{F}[\text{ti} \triangleright q\vartheta] \]

- Trail inspection **reifies** the computation history of the **audited unit** currently being executed
- It allows us to analyse the history by primitive recursion
- Example: \( \vartheta(\beta) = \vartheta(\text{ti}) = 1, \vartheta(t) = \vartheta(\text{app}) = \lambda a, b. a + b, \vartheta(r) = 0 \)

\[

t_0 \leftarrow \iota(\vartheta); \quad \text{Evaluates to 0; emits trails ti and } \beta \\
_{} \leftarrow ((\lambda a, b. (a, b)) 2) 6; \\
t_1 \leftarrow \iota(\vartheta); \\
t_1 - t_0
\]
Auditing is performed by an inspection operator

\[ !q \mathcal{F}[i(\vartheta)] \rightarrow !q \mathcal{F}[\text{ti} \triangleright q\vartheta] \]

Trail inspection reifies the computation history of the audited unit currently being executed.

- It allows us to analyse the history by primitive recursion.
- Example: \( \vartheta(B) = \vartheta(\text{ti}) = 1 \), \( \vartheta(t) = \vartheta(\text{app}) = \lambda a, b. a + b \), \( \vartheta(r) = 0 \)

\[
\begin{align*}
! (t_0 & \leftarrow i(\vartheta)); \quad \text{Evaluates to 0; emits trails ti and B} \\
_ & \leftarrow ((\lambda a, b. (a, b)) 2) 6; \quad \text{Emits trail B 3 times} \\
t_1 & \leftarrow i(\vartheta); \\
t_1 - t_0 &
\end{align*}
\]
Auditing is performed by an *inspection* operator

\[ !_q \mathcal{F}[\iota(\vartheta)] \rightarrow !_q \mathcal{F}[\text{ti} \triangleright q\vartheta] \]

- Trail inspection **reifies** the computation history of the *audited unit* currently being executed
- It allows us to analyse the history by primitive recursion
- Example: \( \vartheta(\beta) = \vartheta(\text{ti}) = 1, \vartheta(t) = \vartheta(\text{app}) = \lambda a, b. a + b, \vartheta(r) = 0 \)

\begin{align*}
! (t_0 & \leftarrow \iota(\vartheta)); & \text{Evaluates to 0; emits trails ti and } \beta \\
_ & \leftarrow ((\lambda a, b. (a, b)) 2) 6; & \text{Emits trail } \beta \text{ 3 times} \\
 t_1 & \leftarrow \iota(\vartheta); & \text{Evaluates to 5, emits trails ti and } \beta \\
t_1 - t_0 & 
\end{align*}
Auditing is performed by an *inspection* operator

\[ !q F[\iota(\vartheta)] \rightarrow !q F[ti \triangleright q\vartheta] \]

- Trail inspection *reifies* the computation history of the *audited unit* currently being executed
- It allows us to analyse the history by primitive recursion
- Example: \( \vartheta(\beta) = \vartheta(ti) = 1, \vartheta(t) = \vartheta(app) = \lambda a, b. a + b, \vartheta(r) = 0 \)

\[
\begin{align*}
! (t_0 & \leftarrow \iota(\vartheta)); && \text{Evaluates to 0; emits trails } ti \text{ and } \beta \\
_ & \leftarrow ((\lambda a, b. \langle a, b \rangle) 2) 6; && \text{Emits trail } \beta \text{ 3 times} \\
t_1 & \leftarrow \iota(\vartheta); && \text{Evaluates to 5, emits trails } ti \text{ and } \beta \\
t_1 - t_0) & \quad \text{Evaluates to 5} 
\end{align*}
\]
The lambda calculus is defined using eager substitution, but functional languages are not implemented that way.

Adapt SECD machine to CAU.
Towards an implementation

- The lambda calculus is defined using eager substitution, but functional languages are not implemented that way.
- Adapt SECD machine to CAU

\[ \lambda M \Rightarrow \lambda qF[q' \Downarrow N] \Rightarrow \lambda t(q,q') F[N] \]

- The main problem with trails is the non-local effect of reduction.
- Why do we need to move the trail to the log, anyway? Let’s *not* do it!
- Can we postpone the operation indefinitely, until we evaluate a trail inspection that asks to read the log?
Anachronisms

\((\lambda x. M x x) (q > N)\)
Anachronisms

\[(\lambda x. M x x) \ (q \triangleright N) \rightarrow \text{app}(r, q) \triangleright (\lambda x. M x x) \ N\]
Anachronisms

\[(\lambda x. M x x) (q \triangleright N) \rightarrow \text{app}(r, q) \triangleright (\lambda x. M x x) N\]

\[\text{app}(r, q) \triangleright \beta \triangleright M N N\]
Anachronisms

\[(\lambda x. M x x) (q \triangleright N) \rightarrow \text{app}(r, q) \triangleright (\lambda x. M x x) N\]

\[\text{app}(r, q) \triangleright \beta \triangleright M N N\]

\[t(\text{app}(r, q), \beta) \triangleright M N N\]
Anachronisms

\[(\lambda x. M x x) \ (q \triangleright N) \rightarrow \text{app}(r, q) \triangleright (\lambda x. M x x) \ N\]

\[\beta \triangleright M(q \triangleright N) \ (q \triangleright N)\]

\[\text{app}(r, q) \triangleright \beta \triangleright M \ N \ N\]

\[t(\text{app}(r, q), \beta) \triangleright M \ N \ N\]
\[(\lambda x. M x x) (q \triangleright N) \rightarrow \text{app}(r, q) \triangleright (\lambda x. M x x) N\]

\[
\beta \triangleright M(q \triangleright N) (q \triangleright N) \quad \text{app}(r, q) \triangleright \beta \triangleright M N N
\]

\[
t(\beta, \text{app}(\text{app}(r, q), q)) \triangleright M N N \quad \text{t(app}(r, q), \beta) \triangleright M N N
\]
Explicit trail projections

- Contribution: CAU$_{\sigma}$ with explicit substitutions and explicit trail projections

\[
M ::= \cdots \mid M[s] \mid [M] \\
q ::= \cdots \mid [M]
\]
Explicit trail projections

- Contribution: CAU$\sigma$ with explicit substitutions and **explicit trail projections**

  \[ M ::= \cdots | M[s] | [M] \]

  \[ q ::= \cdots | [M] \]

- Refined, **time-honouring** computation rules
- Trail-normalisation in micro-steps
- Computation can happen on denormalised terms
Contribution: CAU$^{-}_{\bar{\sigma}}$ with explicit substitutions and explicit trail projections

\[
M ::= \cdots | M[s] | [M]
\]
\[
q ::= \cdots | [M]
\]

- Refined, time-honouring computation rules
- Trail-normalisation in micro-steps
- Computation can happen on denormalised terms

**Theorem:**

1) If \( M \xrightarrow{CAU^{-}} N \) then \( M \xrightarrow{CAU^{-}_{\bar{\sigma}}} N \)

2) If \( M \xrightarrow{CAU^{-}_{\bar{\sigma}}} N \) then \( \sigma\tau(M) \xrightarrow{CAU^{-}} \sigma\tau(N) \)
The usual definition of reduction (computation) is 
**anachronistic** when applied to terms annotated with trails

- These terms have a non-null history which is incorrectly logged 
as a future event

\[(\lambda x. M) \, N \rightarrow \beta \triangleright M[N/x]\]
The usual definition of reduction (computation) is \textit{anachronistic} when applied to terms annotated with trails.

- These terms have a non-null history which is incorrectly logged as a future event.

\textit{CAU} has a \textbf{time-honouring reduction} which is sound even when applied to terms with a past history.

\[(\lambda x. M) N \rightarrow \beta \succ M[^N/x]\]
A weak-CBV abstract machine

 Tuple under evaluation: \( \tau ::= (q | \overline{M} | e) \equiv q \triangleright [\overline{M}[e]] \)

 Values: \( V ::= q \triangleright !q_1 \ldots !q_n [(\lambda \overline{M})[e]] \)

 State: \( \pi, D \) where \( \pi ::= \overline{\pi}, D ::= \overline{V} \)

 Theorem:

 If \( (\pi, D) \to (\pi', D') \), then \( [\pi, D] \xrightarrow{\text{CAU}_\sigma^-} [\pi', D'] \)
A weak-CBV abstract machine

- Tuple under evaluation: \( \tau ::= (q|\bar{M}|e) \equiv q \triangleright [\bar{M}[e]] \)
- Values: \( V ::= q \triangleright !_{q_1} \ldots !_{q_n} [\lambda \bar{M}][e] \)
- State: \( \pi, D \) where \( \pi ::= \tilde{t}, D ::= \tilde{V} \)

\[
\begin{align*}
(q|\bar{M} \bar{N}|e) &::= \pi, D \Rightarrow \\
(r|\bar{M}|e) &::= (r|\bar{N}|e) :: (q|@|e) :: \pi, D \\
\mathcal{E}[q \triangleright [(\bar{M}\bar{N})[e]]] &\xrightarrow{\text{CAU}_\sigma^-} \mathcal{E}[q \triangleright (r \triangleright [\bar{M}[e]]) (r \triangleright [\bar{N}[e]])] \\
\end{align*}
\]

Theorem:

If \( (\pi, D) \Rightarrow (\pi', D') \), then \([\pi, D] \xrightarrow{\text{\text{CAU}_\sigma^-}} [\pi', D']\)
 Tuple under evaluation: \( \tau ::= (q \mid \overline{M} \mid e) \equiv q \triangleright [\overline{M}[e]] \)

 Values: \( V ::= q \triangleright !q_1 \ldots !q_n [\lambda \overline{M}][e] \)

 State: \( \pi, D \) where \( \pi ::= \vec{\tau}, D ::= \vec{V} \)

**Theorem:**

If \((\pi, D) \rightarrow (\pi', D')\), then \([\pi, D] \xrightarrow{\text{CAU}_\sigma^-} [\pi', D']\)
A weak-CBV abstract machine

- Tuple under evaluation: \( \tau ::= (q|\overline{M}|e) \equiv q \triangleright [\overline{M}[e]] \)
- Values: \( V ::= q \triangleright !_{q_1} \ldots !_{q_n} [(\lambda \overline{M})[e]] \)
- State: \( \pi, D \) where \( \pi ::= \tilde{t}, D ::= \tilde{V} \)

\[
\begin{array}{c}
\varepsilon[q \triangleright [(\lambda \overline{M})[e]]] \xrightarrow{\text{CAU}_\sigma^-} \varepsilon[q \triangleright [(\lambda \overline{M})[e]]]
\end{array}
\]

**Theorem:**

If \( (\pi, D) \rightarrow (\pi', D') \), then \( \llbracket \pi, D \rrbracket \xrightarrow{\text{CAU}_\sigma^-} \llbracket \pi', D' \rrbracket \)
A weak-CBV abstract machine

- Tuple under evaluation: \( \tau ::= (q|\bar{M}|e) \equiv q \triangleright [\bar{M}[e]] \)
- Values: \( V ::= q \triangleright !q_1 \ldots !q_n [(\lambda \bar{M})[e]] \)
- State: \( \pi, D \) where \( \pi ::= \vec{\tau}, D ::= \vec{\bar{V}} \)

Theorem:

If \( (\pi, D) \mapsto (\pi', D') \), then \([\pi, D] \xrightarrow{\text{CAU}_{\vec{\sigma}}} [\pi', D']\)
A weak-CBV abstract machine

- Tuple under evaluation: $\tau ::= (q|\vec{M}|e) \equiv q \triangleright [\vec{M}[e]]$
- Values: $V ::= q \triangleright !q_1 \ldots !q_n [(\lambda \vec{M})[e]]$
- State: $\pi, D$ where $\pi ::= \vec{t}, D ::= \vec{V}$

\[
\begin{align*}
(q|\@|e) &::= \pi, (q' \triangleright c') :: (q'' \triangleright [(\lambda \vec{M})[e]]) :: D \rightarrow \\
(q; \text{app}(q'', q'); \beta|\vec{M}|(r \triangleright c) \cdot e) &:: \pi, D
\end{align*}
\]

$E[q \triangleright (\triangleright [\lambda \vec{M}[e]]) (q' \triangleright c)] \xrightarrow{\text{CAU}_{\sigma}} E[q; \text{app}(q'', q'); \beta \triangleright [\vec{M}[c \cdot e]]$

**Theorem:**

If $(\pi, D) \rightarrow (\pi', D')$, then $[\pi, D] \xrightarrow{\text{CAU}_{\sigma}} [\pi', D']$
Conclusions

- **CAU\(\sigma\)**: CAU with ES and **deferrable trail normalisation**
  - Explicit trail projections
  - Time-honouring reduction
  - Correctly refines CAU

- A weak CBV abstract machine
  - All rules \(O(1)\) except trail inspection
  - Proved correct wrt CAU\(\sigma\)

- Future work: application to the Provenance Inspection Calculus