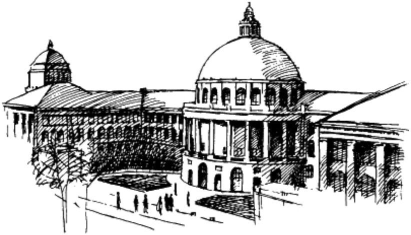
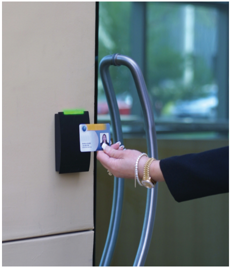
# Enforceable Security Policies

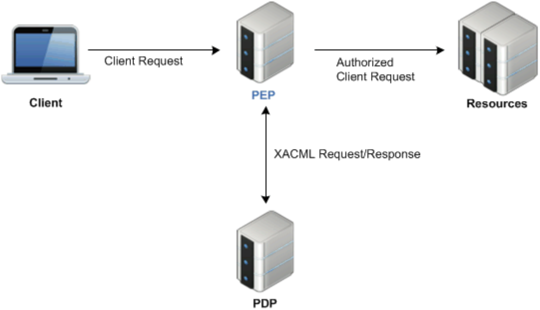
**David Basin**
**ETH Zurich**

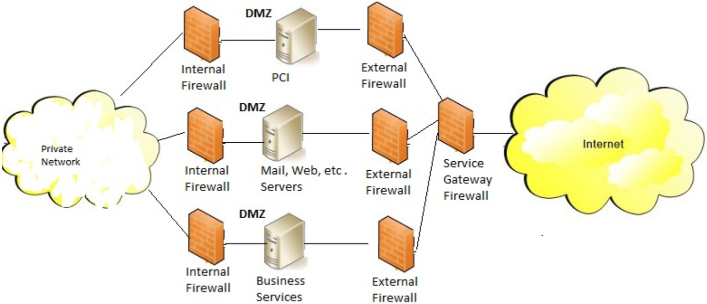


Joint work with Vincent Jugé, Felix Klaedtke and Eugen Zălinescu

# Policy Enforcement Mechanisms are Omnipresent

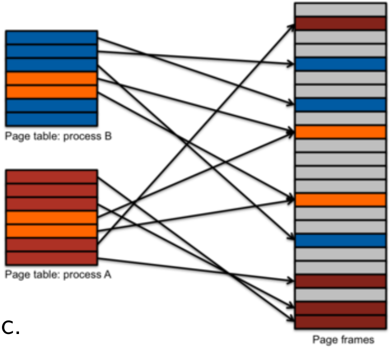




# Enforcing Policies at all Hardware/Software Layers

- Memory management hardware
- Operating systems and file systems
- Middleware and application servers
- Network traffic: firewalls and VPNs
- Applications: databases, mail servers, etc.



# Policies Come in all Shapes and Sizes

History-based Access Control

Chinese Wall

Information Flow

Separation of Duty

Business Regulations

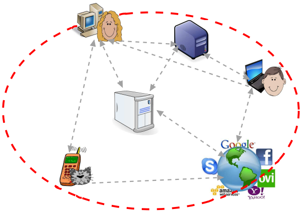
Data Usage

Privacy

...

# So Which Policies can be Enforced?

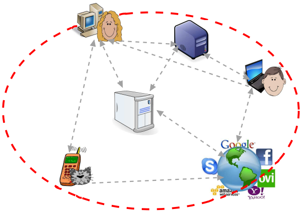
# Examples
# AC / General



policies

- Only **Alice** may update **customer data**.
- **Employees** may overspend their **budget** by 50% provided they previously received **managerial approval**.
- **Bob** may make up to most 5 copies of **movie XYZ**.

# Examples
# AC / General




- Only **Alice** may update **customer data**.
- **Employees** may overspend their **budget** by 50% provided they previously received **managerial approval**.
- **Bob** may make up to most 5 copies of **movie XYZ**.

---

- A **login** must not happen within 3 seconds after a **fail**
- Each **request** must be followed by a **deliver** within 3 seconds

# Relevance of Research Question

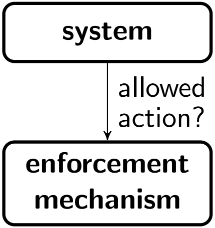

- Fundamental question about **mechanism design**.
  - **Focus:** conventional mechanisms that operate by **monitoring execution** and **preventing** actions that violate policy.
  - Given **omnipresence of such mechanisms** and **diversity of policies** it is natural to ask: **which policies can be enforced?**
- Enforce versus monitor
  - Enforcement often combined with system monitoring.
  - Why do both? Defense in depth? Accountability? Something deeper?
- Fun problem. Nice example of applied theory.
  - Temporal reasoning, logic, formal languages, complexity theory

# Enforcement by Execution Monitoring

*Enforceable Security Policies*
Fred B. Schneider, ACM Trans. Inf. Syst. Sec., 2000

**Abstract Setting**

- System iteratively executes actions
- Enforcement mechanism intercepts them (prior to their execution)
- Enforcement mechanism terminates system in case of violation

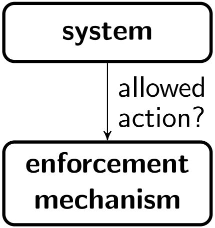


# Enforcement by Execution Monitoring

*Enforceable Security Policies*
Fred B. Schneider, ACM Trans. Inf. Syst. Sec., 2000

## Abstract Setting

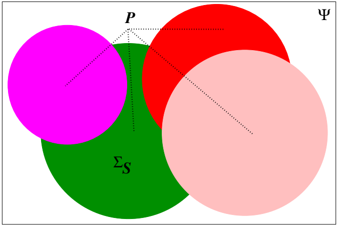
- System iteratively executes actions
- Enforcement mechanism intercepts them (prior to their execution)
- Enforcement mechanism terminates system in case of violation




**So which policies are enforceable?**

# Characterizing EM enforceability — formal setup

- Let $\Psi$ denote universe of all possible finite/infinite sequences.
  - Represents executions at some abstraction level.
  - E.g., sequences of actions, program states, state/action pairs, ...
  - **Example: request · tick · deliver · tick · tick · request · deliver · tick . . .**
- A **security policy** $P$ is specified as a predicate on **sets** of executions, i.e., it characterizes a **subset of** $2^{\Psi}$.
- A system $S$ defines a set $\Sigma_S \subseteq \Psi$ of actual executions.
- $S$ **satisfies** $P$ iff $\Sigma_S \in P$.



# Characterizing EM enforceability: trace properties

- EMs work by monitoring target execution. So any enforceable policy $P$ must be specified so that

$$\Pi \in P \iff \forall \sigma \in \Pi.\, \sigma \in \hat{P}\,.$$

  $\hat{P}$ formalizes criteria used by EM to decide whether a trace $\sigma$ is acceptable, i.e., whether or not to abort ("execution cutting").

- Hence **Requirement 1**: $P$ must be a **property** formalizable in terms of a predicate $\hat{P}$ on executions.

  A set is a **property** iff set membership is determined by each element alone and not by other elements of the set.

- Contrast: properties of **behaviors** vs. properties of **sets of behaviors**.
  * **"Average response time, over all executions, should be $\leq$ 10ms."**
  * **"Actions of high users have no effect on observations of low users."**

# Characterization (cont.)

- Mechanism cannot decide based on possible future execution.

$$\textbf{tick} \cdot \textbf{tick} \cdot \textbf{BadThing} \cdot \textbf{tick} \cdot \textbf{tick} \cdot \textbf{GreatThing} \cdot \textbf{tick} \ldots$$

$$\Uparrow \textbf{???}$$

- Consequence: (Recall $\Pi \in P \Leftrightarrow \forall \sigma \in \Pi.\, \sigma \in \hat{P}$)
  - Suppose $\sigma'$ is a prefix of $\sigma$, such that $\sigma' \notin \hat{P}$, and $\sigma \in \hat{P}$.
  - Then policy $P$ is not enforceable since we do not know whether system terminates before $\sigma'$ is extended to $\sigma$.

- **Requirement 2**, above, is called **prefix closure**.
  - If a trace is not in $\hat{P}$, then the same holds for all extensions.
  - Conversely if a trace is in $\hat{P}$, so are all its prefixes.

- Moreover, **Requirement 3, finite refutability:** If a trace is not in $\hat{P}$, we must detect this based on some finite prefix.

# Characterization (cont.)

- Let $\tau \leq \sigma$ if $\tau$ is a **finite prefix** of $\sigma$.
- **Requirement 2:** prefix closure.

$$\forall \sigma \in \Psi.\, \sigma \in \hat{P} \rightarrow (\forall \tau \leq \sigma.\, \tau \in \hat{P})$$

- **Requirement 3:** finite refutability.

$$\forall \sigma \in \Psi.\, \sigma \notin \hat{P} \rightarrow (\exists \tau \leq \sigma.\, \tau \notin \hat{P})$$

- Sets satisfying all three requirements are called **safety properties**.

# Safety properties — remarks

- **Safety properties** are a class of trace properties.
  Essentially they state that **nothing bad ever happens**.
- **Finite refutability** means if bad thing occurs, this happens after finitely many steps and we can immediately observe the violation.
- **Examples**
  * Reactor temperature never exceeds $1000^o\ C$.
  * If the key is not in the ignition position, the car will not start.
  * You may play a movie at most three times after paying for it.
  * Any history-based policy depending on the present and past.
- **Nonexample** (liveness): If the key is in the ignition position, the car will start eventually.

  **Why?**

# Safety properties — remarks

- **Safety properties** are a class of trace properties.
  Essentially they state that **nothing bad ever happens**.

- **Finite refutability** means if bad thing occurs, this happens after finitely many steps and we can immediately observe the violation.

- **Examples**
  - Reactor temperature never exceeds $1000^{o}\ C$.
  - If the key is not in the ignition position, the car will not start.
  - You may play a movie at most three times after paying for it.
  - Any history-based policy depending on the present and past.

- **Nonexample** (liveness): If the key is in the ignition position, the car will start eventually.

  **Why?** This cannot be refuted on any finite execution.

# Consequences

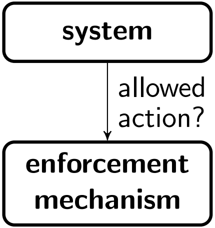
- If set of executions for a security policy $P$ is not a safety property, then no EM enforcement mechanism exists for $P$. **Examples:**
  - Mechanism grants access if a certificate is delivered in future.
  - Some non-trace properties (hyper-properties) like non-interference.
- EM-enforceable policies can be composed by running mechanisms in parallel.
- EM mechanisms can be implemented by automata.
  - Büchi automata are automata on infinite words.
  - A variant, **security automata**, accept safety properties.
    **These constitute a central security model.**
  - Topic of another talk!

# Story so far...

*Enforceable Security Policies*
Fred B. Schneider, ACM Trans. Inf. Syst. Sec., 2000

## Abstract Setting

- System iteratively executes actions
- Enforcement mechanism intercepts them (prior to their execution)
- Enforcement mechanism terminates system in case of violation

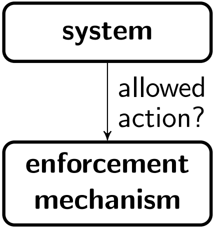


# Story so far...

*Enforceable Security Policies*
Fred B. Schneider, ACM Trans. Inf. Syst. Sec., 2000

## Abstract Setting

- System iteratively executes actions
- Enforcement mechanism intercepts them (prior to their execution)
- Enforcement mechanism terminates system in case of violation




## Main Concerns

- enforceable policy $\overset{\Longrightarrow}{\not\Longleftarrow}$ safety property
- match with reality?

# Follow-Up Work

- *SASI enforcement of security policies*
  Ú. Erlingsson and F. Schneider, NSPW'99
- *IRM enforcement of Java stack inspection*
  Ú. Erlingsson and F. Schneider, S&P'00
- *Access control by tracking shallow execution history*
  P. Fong, S&P'04
- *Edit automata: enforcement mechanisms for run-time security properties*
  J. Ligatti, L. Bauer, and D. Walker, Int. J. Inf. Secur., 2005
- *Computability classes for enforcement mechanisms*
  K. Hamlen, G. Morrisett, and F. Schneider, ACM Trans. Inf. Syst. Secur., 2006
- *Run-time enforcement of nonsafety policies*
  J. Ligatti, L. Bauer, and D. Walker, ACM Trans. Inf. Syst. Secur., 2009
- *A theory of runtime enforcement, with results*
  J. Ligatti and S. Reddy, ESORICS'10
- *Do you really mean what you actually enforced?*
  N. Bielova and F. Massacci, Int. J. Inf. Secur., 2011
- *Runtime enforcement monitors: composition, synthesis and enforcement abilities*
  Y. Falcone, L. Mounier, J.-C. Fernandez, and J.-L. Richier, Form. Methods Syst. Des., 2011
- *Service automata*
  R. Gay, H. Mantel, and B. Sprick, FAST'11
- *Cost-aware runtime enforcement of security policies*
  P. Drábik, F. Martinelli, and C. Morisset, STM'12
- ...

# Match with reality ???

- A **login** must not happen within 3 seconds after a **fail**
- Each **request** must be followed by a **deliver** within 3 seconds

**Both are safety properties.**

**Can we enforce both by preventing events causing policy violations from happening?**

# Some Auxiliary Definitions

- $\Sigma^*$ and $\Sigma^\omega$, are the finite and infinite sequences over alphabet $\Sigma$. $\Sigma^\infty := \Sigma^* \cup \Sigma^\omega$.

- For $\sigma \in \Sigma^\infty$, denote set of its **prefixes** by $\mathsf{pre}(\sigma)$ and set of its **finite prefixes** by $\mathsf{pre}_*(\sigma)$. I.e., $\mathsf{pre}_*(\sigma) := \mathsf{pre}(\sigma) \cap \Sigma^*$.

- The **truncation** of $L \subseteq \Sigma^*$ is the largest prefix-closed subset of $L$.

$$\mathsf{trunc}(L) := \{\sigma \in \Sigma^* \mid \mathsf{pre}(\sigma) \subseteq L\}$$

- Its **limit closure** contains both the sequences in $L$ and the infinite sequences whose finite prefixes are all in $L$.

$$\mathsf{limitclosure}(L) := L \cup \{\sigma \in \Sigma^\omega \mid \mathsf{pre}_*(\sigma) \subseteq L\}$$

- For $L \subseteq \Sigma^*$ and $K \subseteq \Sigma^\infty$, their **concatenation** is defined by:

$$L \cdot K := \{\sigma\tau \in \Sigma^\infty \mid \sigma \in L \text{ and } \tau \in K\}$$

# Refined Abstract Setting Accounting For Controllability

## Actions

**Set of actions** $\Sigma = \mathbf{O} \cup \mathbf{C}$:

- $\mathbf{O} = \{\text{observable actions}\}$
- $\mathbf{C} = \{\text{controllable actions}\}$

## Traces

**Trace universe** $\mathbf{U} \subseteq \Sigma^\infty$:

- $\mathbf{U} \neq \emptyset$
- $\mathbf{U}$ prefix-closed

**Example:** $\mathbf{request} \cdot \mathbf{tick} \cdot \mathbf{deliver} \cdot \mathbf{tick} \cdot \mathbf{tick} \cdot \mathbf{request} \cdot \mathbf{deliver} \cdot \mathbf{tick} \ldots \in \mathbf{U}$

# Refined Abstract Setting Accounting For Controllability

## Actions

**Set of actions** $\Sigma = \mathbf{O} \cup \mathbf{C}$:

- $\mathbf{O}$ = {observable actions}
- $\mathbf{C}$ = {controllable actions}

## Traces

**Trace universe** $\mathbf{U} \subseteq \Sigma^{\infty}$:

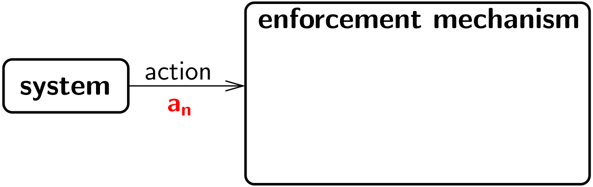
- $\mathbf{U} \neq \emptyset$
- $\mathbf{U}$ prefix-closed

**Example:** $\textbf{request} \cdot \textbf{tick} \cdot \textbf{deliver} \cdot \textbf{tick} \cdot \textbf{tick} \cdot \textbf{request} \cdot \textbf{deliver} \cdot \textbf{tick} \ldots \in \mathbf{U}$

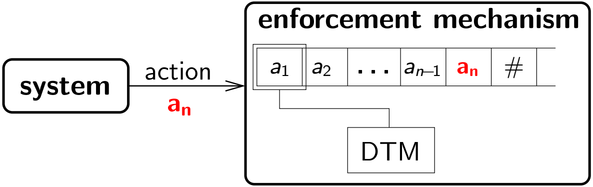
## Requirements (on an Enforcement Mechanism)

- **Soundness**: prevents policy-violating traces
- **Transparency**: allows policy-compliant traces
- **Computability**: makes decisions

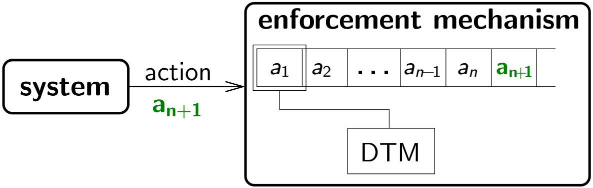
# Formalization



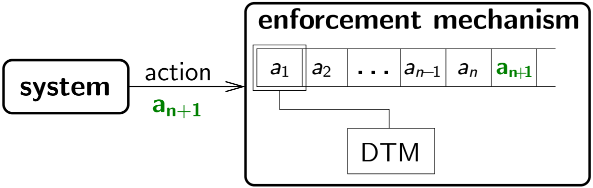
# Formalization



# Formalization



# Formalization




## Definition

$P \subseteq (\mathbf{O} \cup \mathbf{C})^\infty$ is **enforceable** in $\mathbf{U}$ $\overset{\text{def}}{\iff}$ exists DTM $\mathcal{M}$ with

1. $\varepsilon \in L(\mathcal{M})$
   "$\mathcal{M}$ accepts the empty trace"
2. $\mathcal{M}$ halts on inputs in $\big(trunc(L(\mathcal{M})) \cdot (\mathbf{O} \cup \mathbf{C})\big) \cap \mathbf{U}$
   "$\mathcal{M}$ either permits or denies an intercepted action"
3. $\mathcal{M}$ accepts inputs in $\big(trunc(L(\mathcal{M})) \cdot \mathbf{O}\big) \cap \mathbf{U}$
   "$\mathcal{M}$ permits an intercepted observable action"
4. $limitclosure\big(trunc(L(\mathcal{M}))\big) \cap \mathbf{U} = P \cap \mathbf{U}$
   "soundness ($\subseteq$) and transparency ($\supseteq$)"

# Examples

## Setting

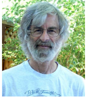
- Controllable actions: $\mathbf{C} = \{\mathbf{login}, \mathbf{request}, \mathbf{deliver}\}$
- Observable actions: $\mathbf{O} = \{\mathbf{tick}, \mathbf{fail}\}$
- Set of actions: $\Sigma = \mathbf{C} \cup \mathbf{O}$
- Trace universe: $\mathbf{U} = \Sigma^* \cup (\Sigma^* \cdot \{\mathbf{tick}\})^\omega$

## Policies

$P_1$. A **login** must not happen within 3 seconds after a **fail**
$\Rightarrow$ **enforceable (TM stops inappropriate login events)**

$P_2$. Each **request** must be followed by a **deliver** within 3 seconds
$\Rightarrow$ **not enforceable (no TM can stop inappropriate tick events)**

# The Evolution of Safety



- L. Lamport, 1977: "A **safety property** is one which states that something bad will *not* happen."

# The Evolution of Safety

- L. Lamport, 1977: "A **safety property** is one which states that something bad will *not* happen."
- B. Alpern and F. Schneider, 1986: A property $P \subseteq \Sigma^\omega$ is $\omega$-**safety** if

$$\forall \sigma \in \Sigma^\omega .\, \sigma \notin P \rightarrow \exists i \in \mathbb{N}.\, \forall \tau \in \Sigma^\omega .\, \sigma^{<i} \cdot \tau \notin P$$

  - Violations are finitely observable and irremedial.
  - Reformulates what we previously saw.

# The Evolution of Safety

- L. Lamport, 1977: "A **safety property** is one which states that something bad will *not* happen."
- B. Alpern and F. Schneider, 1986: A property $P \subseteq \Sigma^\omega$ is **$\omega$-safety** if

$$\forall \sigma \in \Sigma^\omega .\, \sigma \notin P \rightarrow \exists i \in \mathbb{N} .\, \forall \tau \in \Sigma^\omega .\, \sigma^{<i} \cdot \tau \notin P$$

  * Violations are finitely observable and irremedial.
  * Reformulates what we previously saw.

- Folklore: A property $P \subseteq \Sigma^\infty$ is **$\infty$-safety** if

$$\forall \sigma \in \Sigma^\infty .\, \sigma \notin P \rightarrow \exists i \in \mathbb{N} .\, \forall \tau \in \Sigma^\infty .\, \sigma^{<i} \cdot \tau \notin P$$

# The Evolution of Safety

- L. Lamport, 1977: "A **safety property** is one which states that something bad will *not* happen."

- B. Alpern and F. Schneider, 1986: A property $P \subseteq \Sigma^\omega$ is **$\omega$-safety** if

$$\forall \sigma \in \Sigma^\omega.\, \sigma \notin P \rightarrow \exists i \in \mathbb{N}.\, \forall \tau \in \Sigma^\omega.\, \sigma^{<i} \cdot \tau \notin P$$

  - Violations are finitely observable and irremedial.
  - Reformulates what we previously saw.

- Folklore: A property $P \subseteq \Sigma^\infty$ is **$\infty$-safety** if

$$\forall \sigma \in \Sigma^\infty.\, \sigma \notin P \rightarrow \exists i \in \mathbb{N}.\, \forall \tau \in \Sigma^\infty.\, \sigma^{<i} \cdot \tau \notin P$$

- T. Henzinger, 1992: A property $P \subseteq \Sigma^\omega$ is **safety in $\mathbf{U} \subseteq \Sigma^\omega$**

$$\forall \sigma \in \mathbf{U}.\, \sigma \notin P \rightarrow \exists i \in \mathbb{N}.\, \forall \tau \in \Sigma^\omega.\, \sigma^{<i} \cdot \tau \notin P \cap \mathbf{U}$$

# Safety

## (with Universe and Observables)

- Intuition
  - $P$ is safety in **U** *and*
  - Bad things are not caused by elements from **O**.

- Formalization: A property $P \subseteq \Sigma^\infty$ is **(U,O)-safety** if

$$\forall \sigma \in \mathbf{U}.\, \sigma \notin P \rightarrow \exists i \in \mathbb{N}.\, \sigma^{<i} \notin \Sigma^* \cdot \mathbf{O} \wedge \forall \tau \in \Sigma^\infty.\, \sigma^{<i} \cdot \tau \notin P \cap \mathbf{U}$$

  - Generalizes previous defs: $\mathbf{O} = \emptyset$ and $\Sigma^\omega$ and $\Sigma^\infty$ are instances of **U**.
  - As **U** and **O** become smaller it is more likely a trace set $P$ is (**U**,**O**)-safety. (Indeed, for $\mathbf{U} = \emptyset$, $P$ is always (**U**,**O**)-safety).

# Safety

### (with Universe and Observables)

- Intuition
  - $P$ is safety in $\mathbf{U}$ *and*
  - Bad things are not caused by elements from $\mathbf{O}$.

- Formalization: A property $P \subseteq \Sigma^\infty$ is **($\mathbf{U}$,$\mathbf{O}$)-safety** if

$$\forall \sigma \in \mathbf{U}.\, \sigma \notin P \rightarrow \exists i \in \mathbb{N}.\, \sigma^{<i} \notin \Sigma^* \cdot \mathbf{O} \wedge \forall \tau \in \Sigma^\infty.\, \sigma^{<i} \cdot \tau \notin P \cap \mathbf{U}$$

  - Generalizes previous defs: $\mathbf{O} = \emptyset$ and $\Sigma^\omega$ and $\Sigma^\infty$ are instances of $\mathbf{U}$.
  - As $\mathbf{U}$ and $\mathbf{O}$ become smaller it is more likely a trace set $P$ is ($\mathbf{U}$,$\mathbf{O}$)-safety. (Indeed, for $\mathbf{U} = \emptyset$, $P$ is always ($\mathbf{U}$,$\mathbf{O}$)-safety).

- Liveness also generalizes to this setting
  ("something good can happen in $\mathbf{U}$ after actions not in $\mathbf{O}$")

# Example

$P_1$. A **login** must not happen within 3 seconds after a **fail**

$P_2$. Each **request** must be followed by a **deliver** within 3 seconds

- $P_1$ is $\infty$-safety.
  - Any trace that violates $P_1$ has a prefix ending in **login** that violates $P_1$.
  - All extensions of this prefix still violate $P_1$.
- $P_2$ is also $\infty$-safety. Argument analogous with violations due to **tick**.
- But $P_1$ is $(\mathbf{U}, \mathbf{O})$-safety & $P_2$ is not $(\mathbf{U}, \mathbf{O})$-safety, for $\mathbf{O} = \{\mathbf{tick}, \mathbf{fail}\}$
  - $P_1$ violated by executing $\mathbf{login} \in \mathbf{C}$. No policy compliant extensions.
  - For $P_2$ simply consider:

$$\mathbf{request} \cdot \mathbf{tick} \cdot \mathbf{tick} \cdot \mathbf{tick} \cdot \mathbf{tick} \ldots$$

# Safety and Enforceability

**Theorem**

Let $P$ be a property and $\mathbf{U}$ a trace universe with $\mathbf{U} \cap \Sigma^*$ decidable.

$$P \text{ is } (\mathbf{U}, \mathbf{O})\text{-enforceable} \iff \begin{array}{ll} (1) & P \text{ is } (\mathbf{U}, \mathbf{O})\text{-safety}, \\ (2) & \mathrm{pre}_*(P \cap \mathbf{U}) \text{ is a decidable set, and} \\ (3) & \varepsilon \in P. \end{array}$$

Proof uses characterization that

$$P \text{ is } (\mathbf{U}, \mathbf{O})\text{-safety iff limitclosure}(\mathrm{pre}_*(P \cap \mathbf{U}) \cdot \mathbf{O}^*) \cap \mathbf{U} \subseteq P.$$

Schneider's "characterization:" only $\Longrightarrow$ for (1)
where $\mathbf{U} = \Sigma^\infty$ and $\mathbf{O} = \emptyset$

# Realizability of Enforcement Mechanisms

## Fundamental Algorithmic Problems

Given a specification of a policy.

- Is it enforceable?
- If yes, can we synthesize an enforcement mechanism for it?
- With what complexity can we do so?

## Some Results

Deciding if $P$ is (**U**, **O**)-enforceable when both **U** and $P$ are given as

- FSAs is **PSPACE-complete**.
- PDAs is **undecidable**.
- LTL formulas is **PSPACE-complete**.
- MLTL formulas is **EXPSPACE-complete**.

# Checking Enforceability and Safety

**(PDA and FSA)**[1]

## Checking Enforceability

Let **U** and $P$ be given as PDAs or FSAs $\mathcal{A}_{\mathbf{U}}$ and $\mathcal{A}_P$.

1. $\mathrm{pre}_*(L(\mathcal{A}_P) \cap L(\mathcal{A}_{\mathbf{U}}))$ is known to be decidable
2. check whether $\varepsilon \in L(\mathcal{A}_P)$
3. check whether $L(\mathcal{A}_P)$ is $(L(\mathcal{A}_{\mathbf{U}}), \mathbf{O})$-safety

## Checking Safety

Let **U** and $P$ be given as PDAs or FSAs $\mathcal{A}_{\mathbf{U}}$ and $\mathcal{A}_P$.

- PDAs: undecidable in general
- FSAs: generalization of standard techniques

[1] Automata have 2 sets of accepting states, for finite and for infinite sequences.

# Checking Enforceability and Safety

**(LTL and MLTL)**

## Checking Enforceability

Let **U** and $P$ be given as LTL or MLTL formulas $\varphi_{\mathbf{U}}$ and $\varphi_P$.

1. $\mathrm{pre}_*(L(\varphi_P) \cap L(\varphi_{\mathbf{U}}))$ is known to be decidable
2. check whether $\varepsilon \in L(\varphi_P)$
3. check whether $L(\varphi_P)$ is $(L(\varphi_{\mathbf{U}}), \mathbf{O})$-safety

## Checking Safety

Let **U** and $P$ be given as LTL or MLTL formulas $\varphi_{\mathbf{U}}$ and $\varphi_P$.

1. translate $\varphi_{\mathbf{U}}$ and $\varphi_P$ into FSAs $\mathcal{A}_{\mathbf{U}}$ and $\mathcal{A}_P$
2. use the results of the previous slide on $\mathcal{A}_{\mathbf{U}}$ and $\mathcal{A}_P$
3. perform all these calculations on-the-fly

# Beyond a Yes-No Answer

- If **yes** . . .

  synthesize an enforcement mechanism from $\mathcal{A}_P$ and $\mathcal{A}_{\mathbf{U}}$
  (Do so by building FSA security automata for $\mathcal{A}_P \cap \mathcal{A}_{\mathbf{U}}$.)

- If **no** . . .

  return a witness illustrating why $P$ is not $(\mathbf{U}, \mathbf{O})$-enforceable
  (Construct trace in $\mathbf{U} \setminus P$ with suffix in $P$ (violating transparency) or that would not be prevented (violating soundness).)

- If **no** . . .

  return the maximal trace universe $\mathbf{V}$ in which $P$ is $(\mathbf{V}, \mathbf{O})$-enforceable

# Conclusion

## Summary

- Formalization of enforceability in a refined abstract setting
- Characterization of enforceability
- Generalization of notion of safety (and liveness)
- Realizability problem for enforcement
- Interesting connections to control theory (Ramadge-Wonham Framework), not discussed here

## Future Work

- Enforceability for other relevant specification languages
- Tool support for enforcement (PEP/PDP, code weaving, ...).
- How best to combine monitoring and enforcement

# References


- David Basin, Vincent Jugé, Felix Klaedtke and Eugen Zălinescu,
  Enforceable Security Policies Revisited
  *ACM Transactions on Information and System Security*, 2013.

- David Basin, Matúš Harvan, Felix Klaedtke and Eugen Zălinescu,
  Monitoring Data Usage in Distributed Systems,
  *IEEE Transactions on Software Engineering*, 2013.

- David Basin, Matúš Harvan, Felix Klaedtke and Eugen Zălinescu,
  MONPOLY: Monitoring Usage-control Policies,
  *Proceedings of the 2nd International Conference on Runtime Verification (RV)*, 2012.

- David Basin, Ernst-Ruediger Olderog, and Paul Sevinc,
  Specifying and analyzing security automata using CSP-OZ,
  *Proceedings of the 2007 ACM Symposium on Information, Computer and Communications Security (ASIACCS)*, 2007.