Enforcing the GDPR

François Hublet, David Basin, and Srđan Krstić

Institute of Information Security, Department of Computer Science, ETH Zürich,
Zurich, Switzerland {francois.hublet,basin,srdan.krstic}@inf.ethz.ch

Abstract. Violations of data protection laws such as the General Data Protection Regulation (GDPR) are ubiquitous. Currently, building IT support to implement such laws is difficult and the alternatives such as manual controls augmented by auditing are limited and scale poorly. This calls for new automated enforcement techniques that can build on, and enforce, a formalization of the law.

In this paper, we present the first enforceable specification of a core set of GDPR provisions, centered on data-subject rights, and describe an architecture that automatically enforces this specification in web applications. We evaluate our architecture by implementing three case studies and show that our approach incurs only modest development and runtime overhead, while covering the most relevant privacy-related aspects of GDPR that can be enforced at runtime.

Keywords: Data protection · Runtime enforcement · Temporal logic.

1 Introduction

As of May 2023, nearly €3b in fines [13] have been imposed on organizations violating the EU's General Data Protection Regulation (GDPR). Despite these fines, GDPR violations are still ubiquitous [10,32]. In a world where data processing permeates most aspects of life, it is unlikely that ex-post auditing and fines can achieve compliance at scale. Hence, an ex-ante, automated approach to privacy is needed.

Runtime verification (RV) [5] provides a framework for monitoring and enforcing complex properties of computer systems at runtime. In RV, system actions are represented as time-stamped events. System execution then constitutes a trace of events that is checked for compliance with a specification, i.e., a trace property describing the system's desired executions. In particular, online RV tools can check compliance in real-time, with violations being either reported (in the case of *runtime monitoring* [23]) or prevented (in the case of *runtime enforcement* [39]).

The challenge of using RV techniques for automated enforcement of privacy laws is threefold. First, legal provisions must be *faithfully translated* into a formal specification: relevant actions by both software systems and individuals must be encoded as events, and legal provisions turned into a formal specification. Second, the resulting specification must be *enforceable* by an automatic process, also called an enforcer (Section 2.1): one must identify which actions can be suppressed or caused to prevent violations, and then check that all potential violations of the specification can be prevented by such suppression or causation. Third and finally, applications must be *instrumented* so that they emit appropriate events and can suppress or cause relevant actions. In particular, a core requirement of privacy laws is for applications to track information flows in order to faithfully determine where and how personal data is used. Dynamic information-flow control mechanisms can be used to deduce data ownership during execution (Section 2.2).

In this paper, we propose the first comprehensive approach to privacy enforcement that addresses all of the above challenges for a core set of GDPR provisions. Specifically, we leverage an RV approach that uses a logical specification language supporting first-order quantification and (metric) temporal operators, providing both a high degree of expressiveness and real-time enforcement support [24]. The provided expressiveness is crucial for a faithful formalization of GDPR provisions.

We first formalize a core set of GDPR provisions as an enforceable specification (Section 3). Our specification builds on previous work [2] on formalizing and *monitoring* the GDPR using Metric First-Order Temporal Logic (MFOTL) [8]. We show here how GDPR provisions can be *enforced* rather than just monitored. Specifically, we extend the previous work to cover a larger fragment of GDPR provisions, and then use the theory of MFOTL enforcement [24] to show the enforceability of our specification. We cover purpose and consent-based data usage, as well as the rights to access, rectify, erase, restrict, and object to data processing.

We then build an architecture that enforces this specification in web applications (Section 4). Our architecture consists of three components: data subjects' *browsers*, the *WebTTC+ execution environment* where data controllers deploy their applications, and a *privacy platform* to which data subjects delegate the enforcement of their privacy preferences. WebTTC+ is an extension of the recently introduced WebTTC environment (TTC stands for Taint, Track, and Control) for enforcing information-flow policies using taint tracking [25]. WebTTC+ ensures that the applications' behavior complies with privacy laws by interacting with an enforcer deployed at the privacy platform. WebTTC+ reports relevant events to the enforcer, which in turn responds with remedial actions (i.e., sets of events to cause or suppress) that WebTTC+ must execute to ensure compliance. Additionally, the data subjects can directly interact with the privacy platform to manage their consent and exercise their rights. The platform sends events representing data subject queries (e.g., requests for data access or erasure) to the enforcer.

We evaluate our architecture by implementing three web applications: a microblogging app, a conference management system, and a health record manager. We show that our architecture effectively enforces core GDPR provisions while preserving the application's functionality and introducing only a modest overhead (Section 5). In summary, we make the following contributions:

- We propose the first enforceable specification for core GDPR provisions.
- We develop an architecture for enforcing this specification in web applications and show how to instrument relevant actions to obtain appropriate events.
- We evaluate a prototype implementation of our architecture on three case studies. Our approach incurs only modest development and runtime overhead, while covering all relevant GDPR provisions that can be enforced at runtime.

We show that this is the first approach that comprehensively enforces core GDPR provisions (Section 6). The companion repository [26] contains all artifacts.

2 Background

We introduce Metric First-Order Temporal Logic (MFOTL), runtime enforcement, and the WebTTC environment that instruments applications for enforcement.

2.1 Runtime enforcement with Metric First-Order Temporal Logic

Given a specification describing intended system execution, runtime enforcement [39] is the process whereby the system execution is observed by an *enforcer* that detects attempted violations and reacts to *prevent* them. In this paper, we use MFOTL [8] as the specification language and the EnfPoly tool [24] as the enforcer.

Let $\Sigma = (\mathbb{D}, \mathbb{E}, a)$ be a first-order signature, containing an infinite set $\mathbb{D}$ of constant symbols, a finite set of *event names* $\mathbb{E}$, and an arity function $a : \mathbb{E} \to \mathbb{N}$. An *event* is a pair $e(d_1, \ldots, d_{a(e)}) \in \mathbb{E} \times \mathbb{D}^{a(e)}$ of an event name e with arity $a(e)$, and $a(e)$ parameters. Events represent system actions *observable* by the enforcer. Some observable events can be suppressed or caused by the enforcer, while others can only be observed. Hence, $\mathbb{E}$ can be partitioned into sets of suppressable (Sup), causable (Cau), and only-observable (Obs) event names. Given a signature Σ, a *trace* is a sequence of pairs $\sigma = ((\tau_i, D_i))_i$, where the $\tau_i \in \mathbb{N}$ are nondecreasing timestamps and $D_i \in \mathbb{DB}$ is a finite set of events. The empty trace is denoted by ε, the set of traces by $\mathbb{T}$, and a *(trace) property* is any set $P \subseteq \mathbb{T}$. For two traces σ, σ' with σ finite, $\sigma \cdot \sigma'$ is their concatenation, and $|\sigma'|$ is the length of σ'.

Let $\mathbb{I}$ be the set of intervals over $\mathbb{N}$ and let $\mathbb{V}$ be for a countable set of variables. MFOTL formulae over a signature Σ are defined by the grammar

$$\varphi ::= \; r(t_1, \ldots, t_{a(r)}) \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x.\ \varphi \mid \bullet_I \varphi \mid \bigcirc_I \varphi \mid \varphi \,\mathsf{S}_I\, \varphi \mid \varphi \,\mathsf{U}_I\, \varphi,$$

where $x \in \mathbb{V}$, $t_1, \ldots, t_{a(r)} \in \mathbb{V} \cup \mathbb{D}$, $r \in \mathbb{E}$, and $I \in \mathbb{I}$. We derive Boolean $\top := p \vee \neg p$, $\bot := \neg\top$, $\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi)$, $\varphi \Rightarrow \psi := \neg\varphi \vee \psi$, and temporal operators "once" ($\blacklozenge_I \varphi := \top \,\mathsf{S}_I\, \varphi$) and "always" ($\Box_I \varphi := \neg(\top \,\mathsf{U}_I\, \neg\varphi)$). We also define a shorthand for the "inclusive since" operator as $\varphi \,\hat{\mathsf{S}}_I\, \psi := \varphi \,\mathsf{S}_I\, (\varphi \wedge \psi)$. Temporal operators with no interval explicitly given have the interval $[0, \infty)$ instead. Predicates are formulae of the form $r(t_1, \ldots, t_{a(r)})$. A formula that contains no future temporal operators is called a *past-only* formula. We use $\mathsf{fv}(\varphi)$ for the set of φ's free variables.

A *valuation* is any function $v : \mathbb{V} \cup \mathbb{D} \to \mathbb{D}$ such that $v(d) = d$ for all $d \in \mathbb{D}$. We write $v[x \mapsto d]$ for the function equal to v, except for $v(x) = d$. Given a trace $\sigma = ((\tau_i, D_i))_i$, a timepoint $1 \leq i \leq |\sigma|$, a valuation v, and a formula φ, the satisfaction relation $\models$ is defined in Figure 1. We write $v \models_\sigma \varphi$ for $v, 1 \models_\sigma \varphi$ and $\{\sigma \mid \exists v.\ v \models_\sigma \varphi\}$ for the trace property P_φ defined by the formula φ. We focus on closed MFOTL formulae, which are formulae φ where $\mathsf{fv}(\varphi) = \emptyset$.

Trace properties may be defined in terms of infinite traces, whereas enforcers can only observe a finite (prefix of a) trace. Intuitively, a trace must be checked 'prefix-wise', i.e., by evaluating its prefixes in increasing order. A trace complies with a property iff an enforcer for this property accepts all of its prefixes.

An enforceable property must contain the empty trace, i.e., the system must initially comply with the property. Additionally, for any extension $\sigma \cdot (\tau, D)$ of a (non-violating) prefix σ, the enforcer must have enough information to *decide* on

$$
\begin{array}{lll|ll}
v,i \models_\sigma \neg\varphi & \text{iff } v,i \not\models_\sigma \varphi & & v,i \models_\sigma r(t_1,\ldots,t_n) & \text{iff } r(v(t_1),\ldots,v(t_n)) \in D_i \\
v,i \models_\sigma \exists x.\ \varphi & \text{iff } v[x \mapsto d], i \models_\sigma \varphi \text{ for } d \in \mathbb{D} & & v,i \models_\sigma \varphi \lor \psi & \text{iff } v,i \models_\sigma \varphi \text{ or } v,i \models_\sigma \psi \\
\end{array}
$$

$$v,i \models_\sigma \bullet_I \varphi \quad \text{iff } i > 1 \text{ and } v,i-1 \models_\sigma \varphi \text{ and } \tau_i - \tau_{i-1} \in I \quad | \quad v,i \models_\varepsilon \varphi$$

$$v,i \models_\sigma \bigcirc_I \varphi \quad \text{iff } i+1 \leq |\sigma| \text{ and } v,i+1 \models_\sigma \varphi, \text{ and } \tau_{i+1} - \tau_i \in I$$

$$v,i \models_\sigma \varphi\, \mathsf{S}_I\, \psi \quad \text{iff } v,j \models_\sigma \psi \text{ for some } j \leq i,\ \tau_i - \tau_j \in I, \text{ and } v,k \models_\sigma \varphi \text{ for all } k \in (j,i]$$

$$v,i \models_\sigma \varphi\, \mathsf{U}_I\, \psi \quad \text{iff } v,j \models_\sigma \psi \text{ for some } |\sigma| \geq j \geq i,\ \tau_j - \tau_i \in I \text{ and } v,k \models_\sigma \varphi \text{ for all } k \in [i,j)$$

Fig. 1: MFOTL semantics

its compliance with the property. In MFOTL, this means that the formula should not depend on future information in the trace; for example, it can be a past-only formula. Furthermore, there must exist finite sets D^- and D^+ of suppressable and causable events respectively that the enforcer can respectively suppress and cause to ensure satisfaction of the property at the new time point, i.e., $\sigma \cdot (\tau, D \backslash D^- \cup D^+)$ satisfies the property. In general, the problem of checking whether an MFOTL formula defines an enforceable property is undecidable [24]. Instead, we can consider a syntactic fragment, called *guarded* MFOTL (GMFOTL) [24], where each formula is guaranteed to define an enforceable property:

$$\psi ::= \bot \mid s(t_1,\ldots,t_n) \mid \neg c(t_1,\ldots,t_n) \mid \psi \land \varphi \mid \psi \lor \psi \mid \exists x.\ \psi.$$

Here, $s \in \mathsf{Sup}, c \in \mathsf{Cau}$, and φ is any MFOTL formula. When $\mathsf{Sup} \cap \mathsf{Cau} = \emptyset$, GMFOTL is expressively complete [24], i.e., any MFOTL formula that defines an enforceable property can be rewritten to an equivalent GMFOTL formula:

Theorem 1. *Let φ be a past-only closed* MFOTL *formula φ and* $\mathsf{Sup} \cap \mathsf{Cau} = \emptyset$. *Then $P_{\Box\varphi}$ is enforceable iff there exists a* GMFOTL *formula ψ with $\Box\varphi \equiv \Box\neg\psi$.*

EnfPoly [24] is an implementation of an MFOTL enforcer. It takes an GMFOTL formula and a trace, which it incrementally processes. At each processing step it reports sets of events to be suppressed or caused to satisfy the formula.

2.2 The WebTTC execution environment

WebTTC is an execution environment for web applications that can enforce information-flow policies specified as MFOTL formulae [25]. Such formulae are expressed over *information-flow traces* whose events capture inputs, outputs, and the influence of inputs on outputs. For example, the policy "Alice's inputs must not influence outputs to any user for marketing purposes" is formalized in WebTTC as

$$\Box\, [\forall o, ds.\ \mathtt{Out}(ds, \text{Marketing}, o) \Rightarrow \neg(\exists i.\ \mathtt{Itf}(i,o) \land (\blacklozenge\, \mathtt{In}(\text{Alice}, i)))]\,.$$

Here, $\mathtt{Out}(u, \text{Marketing}, o)$ means "the output with identifier o is performed to user ds for marketing purposes", $\mathtt{In}(\text{Alice}, i)$ means "Alice inputs the input with identifier i", and $\mathtt{Itf}(i,o)$ means "the input with identifier i *interferes* with the output with identifier o." The $\mathtt{Itf}$ event captures (non-)interference between inputs and outputs as in the information-flow control (IFC) literature [22].

To enforce such information-flow policies, WebTTC follows a dynamic approach that uses a policy enforcement point (PEP) that supervises application execution together with a policy decision point (PDP) implemented by EnfPoly

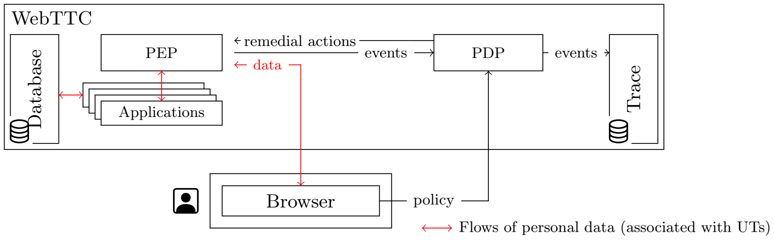


Fig. 2: WebTTC architecture

(see Figure 2). Applications written in a Python-like programming language with an IFC semantics [25] (provided by WebTTC) are deployed inside WebTTC where they interact with a database. All inputs and outputs are observable by the PDP, which can suppress outputs to ensure compliance with a user-specified information-flow policy. User inputs to applications are processed in three steps:

Taint: The PEP tags a new input from a user u with a fresh input identifier, called a *unique taint* (UT), and sends an event $\texttt{In}(u, ut)$ to the PDP, which stores it in the trace. The input and its UT are forwarded to the application.

Track: The semantics of the WebTTC-provided programming language propagates UTs during execution. Each data item in memory or in the database is always tagged with the UTs of all inputs that interfered with its current value.

Control: WebTTC requires that every application output be labeled with a purpose. When an application attempts to output to a user u for some purpose p, the PEP generates a fresh output identifier o as well as events $\texttt{Out}(u, p, o)$ and $\texttt{Itf}(ut, o)$ for each UT ut tagging the output value. It then sends these events to the PDP, which instructs the PEP to suppress the output if it would cause a policy violation. If the output is not suppressed, the trace is updated.

To summarize, WebTTC uses the following first-order signature Σ_{TTC}:

Event	**Description**	**Type**
$\texttt{In}(u, ut)$	Data subject ds inputs a value with UT ut	
$\texttt{Out}(u, p, ut)$	Value with UT ut is output to data subject ds for purpose p	Ⓢ
$\texttt{Itf}(ut, o)$	Input with UT ut interferes with output o	

Ⓢ = Suppressable; unmarked events are only-observable

3 Formalizing GDPR provisions

We now review the core GDPR provisions (Section 3.1), present a first-order signature (Section 3.2), and an MFOTL formula formalizing these core GDPR provisions (Section 3.3). Finally, we show that the formula is enforceable (Section 3.4).

3.1 Core GDPR provisions

We focus on the provisions laid out in Chapters 2 and 3 of the GDPR, which define the *principles* of data processing and the *rights* of the data subject (DS):

– **Purpose and consent-based usage** [Art. 5(1)(b), 6(1), 7(1,3), 9(1,2)]. The processing of personal data is lawful only if the DS has consented to processing *for one or more specific purposes*, unless the data controller can claim a specific legal ground. The DS can revoke consent at any time. Specific legal grounds are applicable in the case of sensitive data (*special data categories*, Art. 9).

- **Right to access** and **right to data portability** [Art. 15(1), 20(1)]. A DS has the right to access any personal data relating to them. They have the right to obtain a copy of this data in a machine-readable format.
- **Right to rectification** [Art. 5(1)(d), Art. 16]. A DS has the right to rectify any inaccurate personal data relating to them.
- **Right to erasure ("right to be forgotten")** [Art. 17(1,2)]. A DS has the right to obtain the erasure of any personal data relating to them. Other controllers with whom the data has been shared shall be notified about it.
- **Right to restriction of processing** [Art. 18(1)]. A DS has the right to restrict the processing of any personal data relating to them.
- **Right to object** [Art. 21(1)]. A DS has the right to object to the processing of data relating to them based on specific legal grounds.

For simplicity, we focus on the case where the DS interacts with applications, each managed by a single data controller. We then formalize the above provisions for each individual application, rather than for each data controller.

3.2 First-order signature for core GDPR concepts

We encode key GDPR concepts as a first-order signature representing observable system actions. The signature (Table 1a) consists of two sets of events. The first set represents actions triggered by a DS: consent (`DSConsent`) and revocation (`DSRevoke`) thereof; requests for accessing (`DSAccess`), deleting (`DSErase`), rectifying (`DSRectify`), or restricting the use of data (`DSRestrict`); repealing restrictions (`DSRepeal`); and objections to processing (`DSObject`). Each event in this set has a name prefixed with `DS`. The second set represents actions triggered by the application: collection (`Collect`), usage (`Use`), and sharing of data (`ShareWith`); granting access to (`GrantAccess`), erasure (`Erase`), and rectification of data (`Rectify`); notifications of data deletion to third-party controllers (`NotifyErase`); and claims of legal grounds to process data (`LegalGround`). All predicates refer to single data items by their unique taints (UTs). The `Collect` (resp. `LegalGrounds`) event has a Boolean parameter *sp* that is `true` if and only if the data collection (resp. the claim of legal grounds) involves (resp. applies to) 'special' data. We call *sp* the *special-data flag* of the data or claim.

DS-triggered actions are not controlled by the enforcer, and hence the corresponding events are only-observable. The collection of data, sharing, and claims for legal grounds, which are not constrained by any core GDPR provision, are also only-observable. In contrast, the usage of data (`Use`) is subject to restrictions and must therefore be suppressable (🅢). Finally, `GrantAccess`, `Erase`, `Rectify` and `NotifyErase` events, which a DS must be able to request, must be causable (🅒).

3.3 MFOTL formula formalizing core GDPR provisions

The core GDPR provisions are formalized by the formula $\varPhi$ given in Table 1b, which is a past-only formula of the form $\Box\neg(\varphi_{\mathrm{Purp}} \lor \cdots \lor \varphi_{\mathrm{Obj}})$ with seven

Table 1: MFOTL formalization of core GDPR provisions

(a) MFOTL signature

Predicate	**Description**	**Type**
	The data subject ds:	
$\texttt{DSConsent}(ds, prp, ut)$	gives consent to use data with UT ut for purpose prp	
$\texttt{DSRevoke}(ds, prp, ut)$	revokes consent given to use data with UT ut for purpose prp	
$\texttt{DSAccess}(ds, ut)$	requests access to data with UT ut	
$\texttt{DSErase}(ds, ut)$	requests erasure of data with UT ut	
$\texttt{DSRectify}(ds, ut, val)$	requests rectification of data with UT ut to new value val	
$\texttt{DSRestrict}(ds, ut)$	requests restriction of data with UT ut	
$\texttt{DSRepeal}(ds, ut)$	repeals restriction of data usage for data with UT ut	
$\texttt{DSObject}(ds, ut)$	objects to usage of data with UT ut by application	
	The application:	
$\texttt{Collect}(ds, ut, sp)$	collects data with UT ut of data subject ds^*	
$\texttt{Use}(prp, ut)$	uses data with UT ut for purpose prp	🅢
$\texttt{ShareWith}(ctr, ut)$	shares data with UT ut with third-party controller ctr	
$\texttt{GrantAccess}(ds, ut)$	gives data subject access to data with UT ut	🅒
$\texttt{Erase}(ut)$	erases data with UT ut	🅒
$\texttt{Rectify}(ut, val)$	rectifies data with UT ut to new value val	🅒
$\texttt{NotifyErase}(ctr, ut)$	notifies controller ctr about erasure of data with UT ut	🅒
$\texttt{LegalGround}(grd, ut, sp)$	claims legal ground grd for using data with UT ut^*	

🅒 = Causable; 🅢 = Suppressable; unmarked events are only-observable

$^*sp$ is a *special-data* Boolean flag that is true iff the data belongs to a special category

(b) MFOTL formula

Description	**Formula**
Overall formula	$\Phi = \Box\neg(\varphi_{\text{Purp}} \vee \varphi_{\text{Acc}} \vee \varphi_{\text{Rect}} \vee \varphi_{\text{Er}} \vee \varphi_{\text{Restr}} \vee \varphi_{\text{Obj}})$
Purpose-based usage	$\varphi_{\text{Purp}} = \exists prp, ut, ds, sp.\ \texttt{Use}(prp, ut) \wedge \blacklozenge\,\texttt{Collect}(ds, ut, sp)$ $\wedge\neg\Big(\Big(\neg\texttt{DSRevoke}(ds, prp, ut)\ \hat{\mathsf{S}}\ \texttt{DSConsent}(ds, prp, ut)\Big)$ $\vee (\exists grd.\ \blacklozenge\,\texttt{LegalGround}(grd, ut, sp)))$
Right to access	$\varphi_{\text{Acc}} = \exists ds, ut, sp.\ \Big(\neg\texttt{GrantAccess}(ds, ut)\ \hat{\mathsf{S}}_I\ \texttt{DSAccess}(ds, ut)\Big)$ $\wedge \blacklozenge\,\texttt{Collect}(ds, ut, sp) \wedge \neg\texttt{GrantAccess}(ds, ut)$ $\underline{or}\ \exists ds, ut, sp.\ \texttt{DSAccess}(ds, ut)$ $\wedge \blacklozenge\,\texttt{Collect}(ds, ut) \wedge \neg\texttt{GrantAccess}(ds, ut)$
Right to rectification	$\varphi_{\text{Rect}} = \exists ds, ut, val, sp.\ \Big(\neg\texttt{Rectify}(ds, ut, val)\ \hat{\mathsf{S}}_I\ \texttt{DSRectify}(ds, ut, val)\Big)$ $\wedge \blacklozenge\,\texttt{Collect}(ds, ut, sp) \wedge \neg\texttt{Rectify}(ds, ut, val)$ $\underline{or}\ \exists ds, ut, sp.\ \texttt{Rectify}(ds, ut, val)$ $\wedge\ \blacklozenge\,\texttt{Collect}(ds, ut, sp) \wedge \neg\texttt{Rectify}(ds, ut, val)$
Right to erasure (1)	$\varphi_{\text{Er1}} = \exists ds, ut, sp.\ \Big(\neg\texttt{Erase}(ut)\ \hat{\mathsf{S}}_I\ \texttt{DSErase}(ds, ut)\Big)$ $\wedge \blacklozenge\,\texttt{Collect}(ds, ut, sp) \wedge \neg\texttt{Erase}(ut)$ $\underline{or}\ \exists ds, ut, sp.\ \texttt{DSErase}(ds, ut)$ $\wedge \blacklozenge\,\texttt{Collect}(ds, ut, sp) \wedge \neg\texttt{Erase}(ut)$
Right to erasure (2)	$\varphi_{\text{Er2}} = \exists ctr, ut.\ \texttt{Erase}(ut)$ $\wedge\ \blacklozenge\,\texttt{ShareWith}(ctr, ut) \wedge\ \neg\texttt{NotifyErase}(ctr, ut)$
Right to restriction	$\varphi_{\text{Restr}} = \exists prp, ut, ds, sp.\ \texttt{Use}(prp, ut) \wedge \blacklozenge\,\texttt{Collect}(ds, ut, sp)$ $\wedge \Big(\neg\texttt{DSRepeal}(ds, ut)\ \hat{\mathsf{S}}\ \texttt{DSRestrict}(ds, ut)\Big)$
Right to object	$\varphi_{\text{Obj}} = \exists prp, ut, ds, sp.\ \texttt{Use}(prp, ut) \wedge \blacklozenge\,\texttt{Collect}(ds, ut, sp)$ $\wedge \blacklozenge\,(\texttt{DSObject}(ds, ut) \wedge \blacklozenge\,(\exists grd.\ \texttt{LegalGround}(grd, ut, sp)))$

I denotes any interval $[i \text{ days}, \infty)$ with $i \leq 30$.

subformulae that describe the *violations* of the provisions in Section 3.1.

Provisions restricting data usage. The formulae $\varphi_{\text{Purp}}, \varphi_{\text{Restr}}$, and φ_{Obj} respectively capture violations of purpose-based usage, the right to restrict processing, and the right to object. Purpose-based usage is violated when the following hold:

1. The data with UT ut is used for purpose prp, which is exactly $\mathtt{Use}(prp, ut)$;
2. The data with UT ut was collected from the DS ds with the special-data flag sp, which is $\blacklozenge\,\mathtt{Collect}(ds, ut, sp)$;
3. (a) Neither the DS ds has given consent to process data with UT ut for purpose prp and has not revoked that consent since then, which is exactly $\neg\mathtt{DSRevoke}(ds, prp, ut)\ \hat{\mathsf{S}}\ \mathtt{DSConsent}(ds, prp, ut)$,
   (b) nor has the application claimed a legal ground grd to use data with UT ut with the special-data flag sp, i.e., $\exists grd.\ \blacklozenge\,\mathtt{LegalGround}(grd, ut, sp)$.

The corresponding formula φ_{Purp} is shown in Table 1b. The formula φ_{Restr} is similar, but does not allow claims of legal grounds. The formula φ_{Obj} formalizes the right to object. This right is violated when all of the following hold:

1. The data with UT ut is used for purpose prp;
2. The data with UT ut was collected from the DS ds;
3. At some time in the past:
   (a) the application has claimed legal ground grd to use data with UT ut,
   (b) and meanwhile, the DS ds has objected to the use of data with UT ut.

We formalize 3. as $\blacklozenge\,(\mathtt{DSObject}(ds, ut) \land \blacklozenge\,(\exists grd, sp.\ \mathtt{LegalGround}(grd, ut, sp)))$.

Provisions regulating DS requests. The formulae φ_{Acc}, φ_{Rect} φ_{Er1}, and φ_{Er2} respectively capture the rights to access, rectification, and (two types of) erasure. Formulae φ_{Acc}, φ_{Rect}, and φ_{Er1} share a similar structure, which requires that the application performs some action in a timely manner when the DS requests it (a so-called *response pattern* [17]). We describe erasure in detail.

According to Art. 12(3) GDPR, erasure must be performed "*without undue delay and in any event within one month of receipt of the request.*" To ensure compliance with this provision, it suffices that an appropriate `Erase` event is caused within 30 days after any `DSErase` event. Both the 'zealous' approach causing `Erase` immediately after `DSErase` and the 'lazy' approach causing it $i \leq 30$ days after `DSErase` are acceptable. These translate into the GMFOTL formulae:

$$\varphi_{\mathrm{Er1,zealous}} = \Box\,\neg\,(\exists ds, ut.\ \neg\mathtt{Erase}(ut) \land \mathtt{DSErase}(ds, ut))$$

$$\varphi_{\mathrm{Er1,lazy},i} = \Box\,\neg\,\Big(\exists ds, ut.\ \neg\mathtt{Erase}(ut)\ \hat{\mathsf{S}}_{[i,\infty)}\ \mathtt{DSErase}(ds, ut)\Big) \qquad \forall i \leq 30.$$

The phrase "*without undue delay*" from Art. 12(3) suggests that the 'zealous' variant is closer to the spirit of the law. As this aspect remains open to interpretation, we show both in Table 1b. The cases of φ_{Rect} and φ_{Acc} are similar.

According to Art. 17(2), erasure of any data item should be accompanied by notifying all controllers with whom this data item was shared. This is formalized by the formula φ_{Er2} in Table 1b. It is violated when some data with UT ut has been shared with a controller at some point in the past ($\blacklozenge\,\mathtt{ShareWith}(pro, ut)$), this data is erased ($\mathtt{Erase}(ut)$), and no notification has been issued ($\neg\mathtt{NotifyErase}(pro, ut)$).

3.4 Enforceability

To use EnfPoly, we must show that formula Φ defines an enforceable property P_{Φ}.

Lemma 1. *$\Psi = \varphi_{\text{Purp}} \lor \varphi_{\text{Acc}} \lor \varphi_{\text{Rect}} \lor \varphi_{\text{Er}} \lor \varphi_{\text{Restr}} \lor \varphi_{\text{Obj}}$ is in GMFOTL.*

Proof. Each disjunct is a GMFOTL formula. The subformulae φ_{Purp}, φ_{Restr}, and φ_{Obj} are all of the form $\exists x_1 \ldots x_k.\ \texttt{Uses}(\ldots) \land \psi$ with ψ past-only, which is a GMFOTL formula since `Uses` is suppressable. The subformulae φ_{Acc}, φ_{Rect}, φ_{Er1}, and φ_{Er2} are all of the form $\exists x_1 \ldots x_k.\ \neg\texttt{X}(\ldots) \land \psi$ or $\exists x_1 \ldots x_k.\ \psi \land \neg\texttt{X}(\ldots)$ where ψ is past-only and `X` is a causable event (`GrantAccess`, `Rectify`, `Erase`, or `NotifyErase`), and are therefore also GMFOTL formulae. Hence formula Ψ, being a disjunction of GMFOTL formulae, is itself a GMFOTL formula.

Since $\Phi = \Box\neg\Psi$ a closed past-only formula, we use Theorem 1 to obtain:

Lemma 2. *P_Φ is enforceable.*

Since Φ is also monitorable [8], we use EnfPoly to enforce it. EnfPoly's algorithm guarantees compliance with the negated disjunction by preventing violations of each disjunct. Violations of φ_{Purp}, φ_{Restr}, and φ_{Obj} are prevented by suppressing `Use` events (i.e., preventing usage), while violations of φ_{Acc}, φ_{Rect}, φ_{Er1}, and φ_{Er2} are prevented by causing `GrantAccess`, `Rectify`, `Erase`, and `NotifyErase`.

4 Enforcement architecture

Figure 3 depicts our enforcement architecture. A DS interacts with applications through their *web browser*, equipped with a browser extension. The extension helps the DS view purposes and set their consent preferences, and sends the corresponding consent events to the PDP. It also allows the DS to declare some inputs as containing special data or personal data of another DS. Finally, the DS can access the platform's privacy dashboard to see how their data has been used.

A controller deploys their applications in the *WebTTC+ execution environment*, which extends WebTTC with support for deletion, rectification, DS access, and notification of other controllers. As in WebTTC, applications' inputs and outputs are controlled by a PEP, which reports critical operations to the PDP and can suppress or cause actions at the PDP's request. However, unlike in WebTTC, the PDP is no longer located inside of WebTTC+, but deployed on a *privacy platform*.

The privacy platform consists of three components: a PDP (as in WebTTC) instantiated with the policy formalized by the MFOTL formula Φ (Section 3.3); a trace (as in WebTTC) storing events of the current application execution; and a *privacy dashboard* [9, 35, 36] used by the DS to query and review events related to their data and exercise their rights. The dashboard can read events from the trace and emit new ones (the `DS` events in Table 1a) on behalf of the DS.

We now explain how the actions corresponding to the events in Section 3.2 are instrumented. We first consider purpose-based usage, the right to restriction, and the right to object, which are enforced through the suppression of `Use` (Section 4.1). Next, we discuss erasure and rectification, whose enforcement requires causation (Section 4.2). Finally, we consider the right to access (Section 4.3).

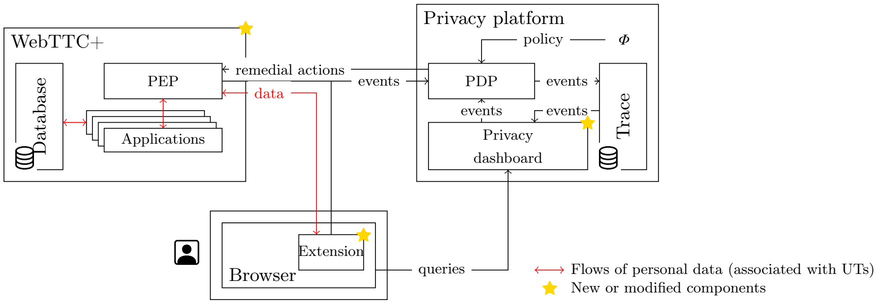


Fig. 3: GDPR enforcement architecture

4.1 Purpose and consent-based usage, restriction, and objection

Data collection and consent are captured by the events `Collect` and `DSConsent`, which are only-observable. The browser extension plays a key role in the data and consent collection process. Recall that it allows the DS to set their privacy preferences (i.e., which data can be processed for which purposes), to declare if some data belongs to a special category, and to declare *whose* personal data they input. An example of the latter is when a physician enters patient data into a software system; the physician would use the extension to declare the patient's identity. Formally, the extension keeps three maps M_{prp}, M_{owners}, and M_{spec} each associating pairs (url, arg) of an URL and argument name to a set of purposes, a set of data subjects, and a special-data flag, respectively. A value sent as argument arg to URL url is considered to belong to $M_{\mathsf{owners}}[(url, arg)]$ data subjects, to contain special data if $M_{\mathsf{spec}}[(url, arg)]$, and can be used for $M_{\mathsf{prp}}[(url, arg)]$ purposes.

Data collection consists of three steps. First, a DS attempts to make a request to an application's URL; the extension retrieves the data DS's preferences and immediately logs the relevant consent events to the PDP. Then, the extension modifies the request to add fresh UTs for each input. Finally, the application receives the request and logs `Collect` events for each input. More formally:

1. The DS ds queries the URL url deployed on a WebTTC+ server with arguments a_i and values v_i, for $1 \leq i \leq k$. The extension intercepts the request and tags the new inputs with fresh UTs $(ut_i)_{1 \leq i \leq k}$. It then sends $\texttt{DSConsent}(ds, prp, ut_i)$ to the PDP for all $1 \leq i \leq k$ and $prp \in M_{\mathsf{prp}}[(url, a_i)]$.
2. The extension sends to the application the original request together with the UTs ut_i, the owners $M_{\mathsf{owners}}[(url, a_i)]$, and the special-data flags $M_{\mathsf{spec}}[(url, a_i)]$.
3. Finally, after it receives the new inputs and their UTs, WebTTC+ logs an event $\texttt{Collect}(ds', app, ut_i, sp_i)$ for each $1 \leq i \leq k$, $ds' \in M_{\mathsf{owners}}[(url, a_i)]$, and $sp_i = M_{\mathsf{spec}}[(url, a_i)]$, and waits for the PDP to acknowledge the logging of all events. Only then can the collected data be processed.

Data usage is captured by `Use`. Purpose-based usage, the right to restriction, and the right to object are all enforced through the suppression of `Use` events.

WebTTC+ emits $\texttt{Use}(prp, ut)$ whenever WebTTC would emit $\texttt{Out}(ds, prp, o)$ and $\texttt{Itf}(ut, o)$ simultaneously. Thus, $\texttt{Use}(prp, ut)$ holds whenever the application performs an output for the purpose prp whose value is influenced by input *ut*. Any `Use` event can be suppressed by suppressing the corresponding output. Being based on non-interference, our definition captures not only the usage of the original DS inputs, but also the usage of any data *derived from DS inputs*. Extending the notion of personal data to data derived from user inputs is in line with the GDPR. Namely, the GDPR defines personal data as "*any information relating to an identified or identifiable natural person*" [Art. 4(1)], a definition that generally encompasses both raw and derived data.

At first glance, controlling data usage only at outputs may appear to depart from the GDPR's definition of processing as "*any operation or set of operations which is performed on personal data or on sets of personal data*" [Art. 4(2)]. However, we claim that this approach does not restrict the violations that we can prevent. First, *purpose* is best understood as an attribute of *business processes* [7] that rely on interactions between computer systems and human agents. Such interactions necessarily involve outputs. Moreover, according to the GDPR, data must be "*limited to what is necessary in relation to the purposes for which they are processed*" ('data minimization', Art. 5(1)(c)). The principle of data minimization is unmonitorable [2], and therefore not amenable to runtime enforcement. Now, data which has no influence on outputs can hardly be considered 'necessary' in relation to the purposes of the business process, since interaction with humans is what business processes are designed for. Hence, any violation of purpose-based usage that cannot be captured by observing outputs is also a violation of data minimization, and thus cannot be prevented using runtime enforcement.

Legal grounds can be claimed by WebTTC+ applications through a special instruction `claim_legal_ground`(grd, sp, x). This claims a legal ground *grd* to use the data contained in a term x with special-data flag *sp*. When the instruction is executed by an application *app*, WebTTC+ logs a `LegalGround`(grd, ut, sp) event for each UT *ut* associated with the value that x evaluates to. Note that the system cannot check that these claims are *legally valid*, as such a check cannot be automated. It therefore only logs the events, which can be objected to later on.

DS requests can be sent by the DS for each of their inputs collected by the applications by interacting with the privacy dashboard. These requests are captured by `DSRevoke`, `DSRestrict`, `DSRepeal`, and `DSObject` events sent to the PDP by the privacy dashboard. In addition to using the extension, users can provide consent manually (i.e., emit `DSConsent` events) through the dashboard.

4.2 Erasure and rectification

Erasure and rectification can be requested by users through the privacy dashboard, which emits `DSErase` and `DSRectify` events on their behalf.

WebTTC+ extends WebTTC's with two functions `erase`(ut) and `rectify`(ut, v) that can be caused by the PDP to erase (resp. rectify) data stored in the database. Additionally, WebTTC+ allows applications to declare *handler functions* that can be used to restore a consistent application state in the event of erasure. Assuming a relational database, `erasure`(ut) is implemented as follows:

1. Identify all tables, rows, and fields tagged with *ut*;

2. Perform erasure by deleting the identified table content or rows or setting the identified fields to a default value;
3. For each erased table, row, or field, call the handling function, if it exists;
4. Emit `Erase`(ut).

The case of rectification is slightly more complex. Namely, rectification should behave differently for raw data and derived data. For raw data, the old input value can be immediately replaced with the new input value. Derived data can only be set to a default, after which the application can provide a way for it to be recomputed using the new input value. The algorithm for `rectify`(ut, v) is as follows:

1. Identify all tables, rows, and fields tagged with ut;
2. Erase the content of any identified tables or rows;
3. For any identified field, if it contains raw data, replace its value by v, otherwise, set it to a default value;
4. For each identified table, row, or field, call the handling function, if it exists;
5. Emit `Rectify`(ut, v).

To distinguish raw data from derived data, we extend TTC's memory model with *integrity tags*. Instead of pairs $\langle v, \alpha \rangle$ of a value v and a set of UTs α, our memory model relies on triples $\langle v, \alpha, \beta \rangle$, where β is either $\mathsf{Derived}$ or $\mathsf{Raw}(ut)$, where ut is a UT. Whenever a data item in memory is equal to $\langle v, \alpha, \mathsf{Raw}(ut) \rangle$, its value v must be equal to the original value of the input with UT ut. When it is equal to $\langle v, \alpha, \mathsf{Derived} \rangle$, its value may not be equal to the value of any input.

After erasure, the formula φ_{Er2} triggers notification of all controllers with whom the erased data has been previously shared. WebTTC+ triggers an event `ShareWith`(pro, ut) on each output influenced by ut whose recipient is a third-party controller, rather than a DS. The causable event `NotifyErase` is instrumented by exposing a function `notify`(ut, ctr) that can be triggered by the PDP. This function notifies the third-party controller ctr about the erasure of ut.

4.3 Access to data

The DS requests access to data through the privacy dashboard, which emits `DSAccess` events. On receiving a `DSAccess` event, the PDP instructs the PEP to cause a corresponding `GrantAccess` event. The PEP then calls a function `access`(ds, ut) exposed by WebTTC+, which has the following behavior:

1. Identify all tables, rows, and fields tagged with ut;
2. Copy all identified tables, rows, and fields into a specific `access` table.
3. Provide the DS with a link to an interface that supports browsing, and downloading, a machine-readable dump of the data (for data portability).
4. Emit `GrantAccess`(ds, ut).
5. In the interface, use the standard WebTTC+ mechanism to show *ds only the data that can be shown without violating other data subjects' consent.*

Step 5 resolves the tension arising between the right to access and other rights when several users' inputs are aggregated. When some data d has been produced by combining inputs from users A and B, granting A access to d may violate purpose-based usage from B's viewpoint, hamper B's capacity to erase or rectify d, and interfere with B's right to restrict the processing of d (see [29] for a discussion of the GDPR's ambiguity in matters of shared data ownership). We take a conservative approach that prioritizes consent-based usage over access, subjecting the extracted data to the same enforcement procedures as application outputs.

5 Evaluation

We now evaluate a prototype implementation of our enforcement architecture with regard to its development and runtime overhead and its coverage of GDPR provisions. Our prototype includes the WebTTC+ environment, the privacy platform, and a Firefox extension. Overall it has 5.5k lines of code (LoC) in Python with 2.7k LoC reused from the WebTTC environment from previous work [25]. The WebTTC+ environment and the privacy dashboard use the Flask web framework and SQLite databases. The PDP is based on EnfPoly [24], with additional Python code ensuring synchronization with a QuestDB time-series database storing the log. Appendix B shows screen captures of the privacy dashboard, the browser extension, and one of the deployed case study applications. Our evaluation aims to answer the following research questions:

RQ1: Can realistic web applications be developed in WebTTC+? If yes, how do the additional privacy requirements impact the size of their code base?

RQ2: How much runtime overhead does WebTTC+ incur compared to a baseline application without support for automated privacy enforcement?

RQ3: What share of the GDPR's provisions does our implementation effectively enforce? What aspects are not covered?

To answer the above questions, we port the following applications:

- **Minitwit**$^{+}$, a clone of the microblogging **Minitwit** application [46];
- **Conf**, a conference management system [49]; and
- **HIPAA**, a health record management system [49].

These applications have been previously used to evaluate the performance of various IFC frameworks [25,30,34,46,49]. We use the variants of these applications presented in Hublet et al. [25] where every output is labeled with a purpose in $\{\texttt{Service}, \texttt{Analytics}, \texttt{Marketing}\}$, and compare our WebTTC+ implementation to a baseline implementation in Python/Flask with the same database backend.

RQ1: Development effort. We have implemented all three applications in our framework, preserving their original functionality. Most of the code from the baseline implementation has been reused with only minor changes. This makes porting Flask applications to WebTTC+ straightforward. Privacy-specific code (e.g., the handling functions) accounts for less than 10% of all application code in the WebTTC+ implementations. We show examples of such code code in **Minitwit**$^{+}$ in Appendix A. Table 2 shows the number of LoC for each case study.

RQ2: Performance overhead. We compare the runtime performance of WebTTC+ with a Python/Flask baseline implementation. Since our baseline does not enforce privacy, this yields an upper bound on the overhead of GDPR enforcement for web apps. We measure the latency of the following representative workloads:

- In **Minitwit**$^{+}$, we show 30 TIMELINE messages and post a NEW MESSAGE;

Application	WebTTC+			Python/Flask	
	Functionality	Privacy	Template	Functionality	Template
Minitwit$^+$	140	16	100	121	119
Conf	312	47	502	284	508
HIPAA	142	25	852	136	846

Table 2: Lines of code of the WebTTC+ and the baseline implementations

- In **Conf**, we show ALL PAPERS and ONE PAPER, and SUBMIT a paper;
- In **HIPAA**, we show ONE PATIENT and ALL PATIENTS.

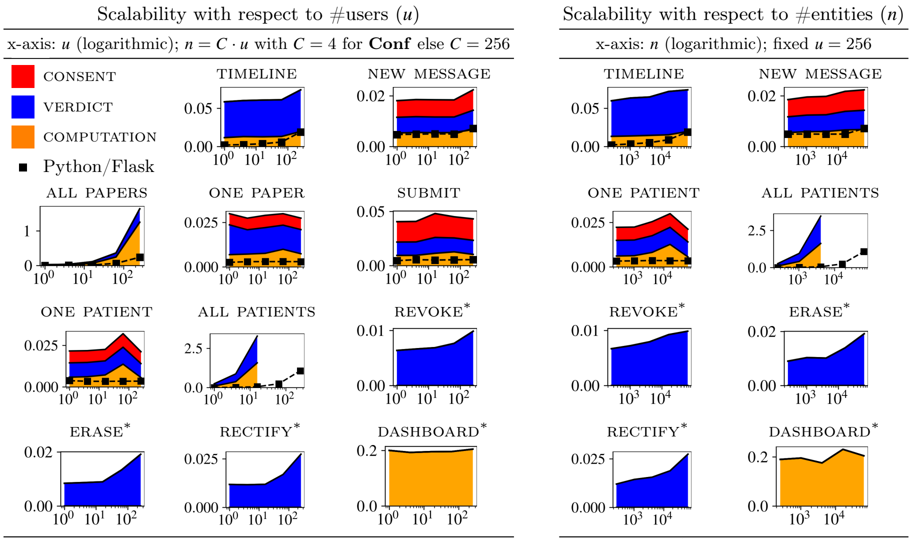
The ALL PAPERS and ALL PATIENTS workloads constitute stress tests [49] used to measure the runtime behavior of applications in the presence of large outputs. In real production scenarios, showing all entities stored in a system is generally avoided and pagination is used to improve runtime performance.

For each workload, we measure the time spent on (1) registering CONSENT at the PDP, (2) waiting for the PDP's VERDICTs, and (3) COMPUTATION within WebTTC+. Additionally, we measure the latency of revoking consent (REVOKE), erasing an input (ERASE) or rectifying it (RECTIFY), and opening the privacy DASHBOARD. We perform the measurements on a high-end laptop (Intel Core i5-1135G7, 32 GB RAM) over $N = 100$ repetitions, while varying the number u of users and the number n of entities (messages for **Minitwit$^+$**, papers for **Conf**, and patients for **HIPAA**) in the database. The results are shown in Figure 4.

For all workloads displaying a constant number of entities (i.e., TIMELINE, NEW MESSAGE, ONE PAPER, SUBMIT, ONE PATIENT, REVOKE, ERASE, and RECTIFY), our architecture incurs an overhead of at most one order of magnitude with respect to the Flask/Python baseline. Executing applications in WebTTC+ (i.e., COMPUTATION) adds at most 10 ms of extra latency with respect to the baseline. The PDP is the main performance bottleneck: WebTTC+ spends 70%–90% of its running time waiting for PDP verdicts, whose latency grows logarithmically. The total latency however remains below 75 ms, allowing for the seamless usage of the applications. For the stress-test workloads showing a number of entities linear in n (ALL PAPERS and ALL PATIENTS), runtime performance remains within one order of magnitude of Flask/Python, with logarithmic growth. Comparable runtime performance was obtained in previous studies using the same applications [25, 49].

RQ3: GDPR coverage. Beyond articles 5(1)(b,d), 6(1), 7(1,3), 9(1,2), 13(1), 15(1), 20(1), 16, 17(1,2), 18(1), and 21(1), our implementation provides at least a technical starting point for enforcing the following provisions:

- **Lawfulness** [Art. 5(1)(a)];
- **Transparency** [Art 12(1,3)]. Any DS-requested information shall be provided in transparent form, without undue delay and within at most one month;
- **Right to information** [Art. 13(1), 15(1)]. A DS has a right to be informed when their personal data is collected and processed;
- **Privacy by design** [Art. 25(1)]. Appropriate technical measures shall be implemented to provide privacy by design;
- **Record of processing activities** [Art. 30(1)]. Data controllers shall keep a record of all processing activities conducted under their responsibility.



*Figures computed with the system loaded with **Minitwit**$^+$ data

Fig. 4: Workload latency (y-axis: seconds per request, avg. over $N = 100$ requests)

Together, articles 5(1)(a,b,d), 6-7, 9, 12-13, 15-18, 21, 25, and 30 appear in 66% of the violations reported on `www.enforcementtracker.com` (May 2023). While this only provides a rough estimate of the (maximal) coverage that can be expected from our approach, it also clearly underlines the potential of runtime enforcement techniques. The following non-covered provisions are cited in at least 5% of the cases: Art. 32 (security of processing, 22%); Art. 5(1)(c) (data minimization, 10%), and Art. 5(1)(f) (storage limitation, 9%). Security is a concern orthogonal to ours, whereas data minimization and storage limitation are generally not monitorable [2].

6 Related work

GDPR formalization. Our formalization of a core of GDPR is closest to Arfelt et al.'s work on monitoring the GDPR [2]. We extend their formalization by introducing UTs as input identifiers and covering special data categories and the right to erasure. We also fix two inaccuracies that we report in Appendix C.

Robaldo et al. have formalized a large part of the GDPR using reified Input/Output logic [37,38] and validated their formalization with legal experts [6]. Their work focuses on accurately encoding the law, but does not support enforcement. Smaller GDPR fragments have been represented in formalisms with some support for automated reasoning, such as deontic logic [1,31], LegalRuleML [33], OCL [44], OWL2 [11], and Prolog [16]. In a different line of research, several policy languages were designed explicitly with GDPR provisions in mind [3,20,41,47,48].

GDPR compliance by design. Many approaches enforce a specific GDPR provision in software systems. To the best of our knowledge, none of these relies on a logical

						Right to...						
		Consent	Purpose	Legal grounds	Special data	access	rectification	erasure	restriction	object	Derived data	Implementation
Language-based	This work	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Ferrara & Spoto [18]	✓	✓								✓	
	Ferreira et al. [19]	✓	✓								✓	✓
	Hublet et al. [25]	✓	✓								✓	✓
	Karami et al. [28]	✓	✓					✓				
	Tokas et al. [43]	✓	✓			✓					✓	
	Wang et al. [47, 48]	✓	✓								✓	✓

						Right to...						
		Consent	Purpose	Legal grounds	Special data	access	rectification	erasure	restriction	object	Derived data	Implementation
Data-store-based	Barati et al. [4]	*	✓									✓
	Chhetri et al. [12]	*	✓									
	Dauden et al. [14]	*	✓			*	*	*	*	*		✓
	Davari & Bertino [15]	*	✓									
	Gjermundrød et al. [21]	*	✓				*					✓
	Schwarzkopf et al. [40]	✓	✓			✓		✓		✓	✓	
	Truong et al. [45]	*				*	*	*	*			✓

* Provision is covered, but data leaving the data store is not protected.

Fig. 5: GDPR compliance by design: coverage of the provisions in Section 3.1

formalization of the GDPR. Figure 5 shows existing approaches based on their GDPR coverage: *consent*-based usage, *purpose*-based usage, *legal grounds*, *special data* categories; coverage of the *rights to access*, *rectification*, *erasure*, *restriction*, and *objection*; protection of *derived data*; and availability of an *implementation*.

Existing work can be classified into language-based approaches [42], which include TTC [25], and data-store-based approaches [27]. In language-based approaches, a programming language is instrumented to ensure privacy compliance either statically [18, 19, 43, 46, 47] or at runtime [19, 25, 28]. Language-based approaches can typically protect derived data, but they have not been widely applied so far beyond consent-based usage. In data-store-based approaches, personal data is stored in protected database containers (*data stores*) that controllers query through an API. Access-control mechanisms ensure data is only accessed with user consent. Within data stores, most GDPR rights can be exercised seamlessly [14,45]. Data processing, however, happens mainly outside of the data stores; as a result, data that leaves the database – and in particular, derived data – is no longer protected. The only work discussing the protection of derived data with such an approach [40] provides just a high-level roadmap without a concrete implementation.

7 Conclusion

We have presented the first enforceable specification of core GDPR provisions that comes together with an enforcement architecture for web applications. To the best of our knowledge, this work is the first to enforce a core set of GDPR provisions and protect derived data by tracking information flows.

We envision three main directions for future work. First, our coverage of legal provisions can be extended beyond the current fragment by relying on a more comprehensive but still enforceable formalization of the GDPR, which we plan to develop. Second, we will further investigate the performance of our enforcement mechanism under very large data volumes and number of users and study optimizations that may ease its deployment in real-world settings. Finally, we plan to extend our architecture by incorporating complementary techniques that allow for the coverage of those aspects of the GDPR (e.g., data and storage minimization) not readily amenable to monitoring techniques.

Acknowledgments Arduin Brandts contributed to a preliminary version of the enforcement signature presented in Section 3.2. Jonas Degelo contributed to the development of the PDP prototype. Ahmed Bouhoula provided feedback on a earlier draft of the paper. François Hublet is supported by the Swiss National Science Foundation grant "Model-driven Security & Privacy" (204796).

References

1. Amantea, I.A., Robaldo, L., Sulis, E., Boella, G., Governatori, G.: Semi-automated checking for regulatory compliance in e-health. In: EDOCW 2021. IEEE (2021)
2. Arfelt, E., Basin, D., Debois, S.: Monitoring the GDPR. In: Sako, K., Schneider, S., Ryan, P.Y.A. (eds.) ESORICS 2019. LNCS, vol. 11735. Springer (2019)
3. Baramashetru, C.P., Tapia Tarifa, S.L., Owe, O., Gruschka, N.: A policy language to capture compliance of data protection requirements. In: IFM 2022. Springer (2022)
4. Barati, M., Rana, O., Petri, I., Theodorakopoulos, G.: GDPR compliance verification in Internet of Things. IEEE Access **8** (2020)
5. Bartocci, E., Falcone, Y., Francalanza, A., Reger, G.: Introduction to runtime verification. Lectures on Runtime Verification: Introductory and Advanced Topics (2018)
6. Bartolini, C., Lenzini, G., Santos, C.: A legal validation of a formal representation of GDPR articles. In: JURIX 2018 (2018)
7. Basin, D., Debois, S., Hildebrandt, T.: On purpose and by necessity: compliance under the GDPR. In: FC 2018. Springer (2018)
8. Basin, D., Klaedtke, F., Müller, S., Zălinescu, E.: Monitoring metric first-order temporal properties. JACM **62**(2) (2015)
9. Bier, C., Kühne, K., Beyerer, J.: PrivacyInsight: the next generation privacy dashboard. In: APF 2016. Springer (2016)
10. Bollinger, D., Kubicek, K., Cotrini, C., Basin, D.: Automating Cookie Consent and GDPR Violation Detection. In: USENIX Security 2022 (2022)
11. Bonatti, P.A., Ioffredo, L., Petrova, I.M., Sauro, L., Siahaan, I.R.: Real-time reasoning in OWL2 for GDPR compliance. Artificial Intelligence **289** (2020)
12. Chhetri, T.R., Kurteva, A., DeLong, R.J., Hilscher, R., Korte, K., Fensel, A.: Data Protection by Design Tool for Automated GDPR Compliance Verification Based on Semantically Modeled Informed Consent. Sensors **22**(7) (2022)
13. CMS: GDPR Enforcement Tracker (2023), www.enforcementtracker.com
14. Daudén-Esmel, C., Castellà-Roca, J., Viejo, A., Domingo-Ferrer, J.: Lightweight blockchain-based platform for GDPR-compliant personal data management. In: CSP 2021 (2021)
15. Davari, M., Bertino, E.: Access control model extensions to support data privacy protection based on GDPR. In: BigData 2019. IEEE (2019)
16. De Montety, C., Antignac, T., Slim, C.: GDPR modelling for log-based compliance checking. In: IFIPTM 2019. Springer (2019)
17. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Property specification patterns for finite-state verification. In: FMSP 1998 (1998)
18. Ferrara, P., Spoto, F.: Static Analysis for GDPR Compliance. In: ITASEC (2018)
19. Ferreira, M., Brito, T., Santos, J.F., Santos, N.: RuleKeeper: GDPR-Aware Personal Data Compliance for Web Frameworks. In: S&P 2023. IEEE (2022)
20. Gerl, A., Bennani, N., Kosch, H., Brunie, L.: LPL, towards a GDPR-compliant privacy language: formal definition and usage. Transactions on Large-Scale Data- and Knowledge-Centered Systems XXXVII (2018)

21. Gjermundrød, H., Dionysiou, I., Costa, K.: privacyTracker: a privacy-by-design GDPR-compliant framework with verifiable data traceability controls. In: ICWE 2016 International Workshops. Springer (2016)
22. Goguen, J.A., Meseguer, J.: Security policies and security models. In: S&P 1982. IEEE (1982)
23. Havelund, K., Rosu, G. (eds.): Runtime Verification, ENTCS, vol. 55. Elsevier (2001)
24. Hublet, F., Basin, D., Krstić, S.: Real-time policy enforcement with metric first-order temporal logic. In: ESORICS 2022. vol. II. Springer (2022)
25. Hublet, F., Basin, D., Krstić, S.: User-controlled Privacy: Taint, Track, and Control. In: Proceedings of Privacy Enforcing Technologies (PoPETS) (2024), to appear
26. Hublet, F., Basin, D., Krstić, S.: Companion repository for "Enforcing the GDPR" (2023), https://gitlab.ethz.ch/fhublet/enforcing-the-gdpr
27. Janssen, H., Cobbe, J., Norval, C., Singh, J.: Decentralized data processing: personal data stores and the GDPR. International Data Privacy Law **10**(4) (2020)
28. Karami, F., Basin, D., Johnsen, E.B.: DPL: A Language for GDPR Enforcement. In: CSF 2022. IEEE (2022)
29. Kutyłowski, M., Lauks-Dutka, A., Yung, M.: GDPR–challenges for reconciling legal rules with technical reality. In: ESORICS 2020. vol. I. Springer (2020)
30. Lehmann, N., Kunkel, R., Brown, J., Yang, J., Vazou, N., Polikarpova, N., Stefan, D., Jhala, R.: STORM: Refinement Types for Secure Web Applications. In: OSDI 2021 (2021)
31. Libal, T.: Towards automated GDPR compliance checking. In: TAILOR 2020. Springer (2021)
32. Nguyen, T.T., Backes, M., Marnau, N., Stock, B.: Share First, Ask Later (or Never?)-Studying Violations of GDPR's Explicit Consent in Android Apps. In: USENIX Security (2021)
33. Palmirani, M., Governatori, G.: Modelling legal knowledge for GDPR Compliance Checking. In: JURIX 2018 (2018)
34. Polikarpova, N., Stefan, D., Yang, J., Itzhaky, S., Hance, T., Solar-Lezama, A.: Liquid information flow control. PACMPL **4**(ICFP) (2020)
35. Puhlmann, N., Wiesmaier, A., Heinemann, A.: Privacy Dashboards for Citizens and GDPR Services for Small Data Holders: A Literature Review. arXiv (2023)
36. Raschke, P., Küpper, A., Drozd, O., Kirrane, S.: Designing a GDPR-compliant and usable privacy dashboard. IFIP 2017 (2018)
37. Robaldo, L., Bartolini, C., Palmirani, M., Rossi, A., Martoni, M., Lenzini, G.: Formalizing GDPR provisions in reified I/O logic: the DAPRECO knowledge base. JLLI **29** (2020)
38. Robaldo, L., Sun, X.: Reified input/output logic: Combining input/output logic and reification to represent norms coming from existing legislation. J. Log. Comput. **27**(8) (2017)
39. Schneider, F.B.: Enforceable security policies. TISSEC **3**(1) (2000)
40. Schwarzkopf, M., Kohler, E., Frans Kaashoek, M., Morris, R.: Position: GDPR compliance by construction. In: VLDB 2019 Workshops. Springer (2019)
41. Tokas, S., Owe, O.: A formal framework for consent management. In: FORTE 2020. Springer (2020)
42. Tokas, S., Owe, O., Ramezanifarkhani, T.: Language-based mechanisms for privacy-by-design. Privacy and Identity Management. Data for Better Living (2020)
43. Tokas, S., Owe, O., Ramezanifarkhani, T.: Static checking of GDPR-related privacy compliance for object-oriented distributed systems. JLAMP **125** (2022)

44. Torre, D., Soltana, G., Sabetzadeh, M., Briand, L.C., Auffinger, Y., Goes, P.: Using models to enable compliance checking against the GDPR: an experience report. In: MODELS 2019. IEEE (2019)
45. Truong, N.B., Sun, K., Lee, G.M., Guo, Y.: GDPR-compliant personal data management: A blockchain-based solution. TIFS **15** (2019)
46. Wang, F., Ko, R., Mickens, J.: Riverbed: Enforcing user-defined privacy constraints in distributed web services. In: NSDI 2019 (2019)
47. Wang, L., Khan, U., Near, J., Pang, Q., Subramanian, J., Somani, N., Gao, P., Low, A., Song, D.: PrivGuard. Privacy Regulation Compliance Made Easier. In: USENIX Security 2022 (2022)
48. Wang, L., Near, J.P., Somani, N., Gao, P., Low, A., Dao, D., Song, D.: Data capsule: A new paradigm for automatic compliance with data privacy regulations. In: VLDB 2019 Workshops. Springer (2019)
49. Yang, J., Hance, T., Austin, T.H., Solar-Lezama, A., Flanagan, C., Chong, S.: Precise, dynamic information flow for database-backed applications. In: Krintz, C., Berger, E. (eds.) PLDI 2016 (2016)

A Privacy code in Minitwit

In **Minitwit**$^{+}$, the additional code can be summarized as follows:

- In functions displaying user messages, we use a function filter_check to select only those messages for which the DS have given consent for purpose `Service`.

  ```
  def filter_check(messages):
      messages2 = []
      checks = check_all('Service', messages)
      for i in range(0, len(messages)):
          if checks[i]:
              messages2.append(messages[i])
      return messages2
  ```

  The function **check_all** is provided by WebTTC+. It takes a purpose p and a list of pairs of values and sets of UTs $[\langle v_1, \alpha_1\rangle, \ldots, \langle v_k, \alpha_k\rangle]$, and returns a list of Booleans $[b_1, \ldots, b_k]$ such that each b_i is true iff v_i can be used with purpose p. To obtain the b_i, WebTTC+ communicates with the PDP [25].
- We define handlers to support the deletion and rectification of messages and friendship relationships. Deleting the text of the message or the ID of the friend should trigger the deletion of the entire message or relation, while messages whose text has been rectified should be marked as `[edited]`:

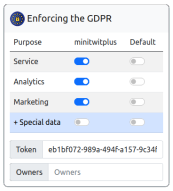
  ```
  @handle_field_deletion('message', 'text')
  def handle_message_text_deletion(i):
      sql("DELETE␣FROM␣message␣WHERE␣id␣=␣?0", [i])
      return None

  @handle_field_rectification('message', 'text')
  def handle_message_text_rectification(i, new_text):
      row = sql("""SELECT author_id, text, pub_date
                   FROM message WHERE id = ?0""", [i])[0]
  ```

```
    sql("""INSERT INTO message (author_id, text, pub_date)
            VALUES (?0, ?1, ?2)""",
        [row[0], new_text + " [edited]", row[2]])
    sql("DELETE FROM message WHERE id = ?0", [i])
    return None
```

The correctors **handle_field_deletion** and **handle_field_rectification**, provided by WebTTC+, allow programmers to define handlers.

B User interface

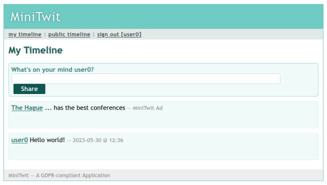


(a) The browser extension

Enforcing the GDPR

Purpose	minitwitplus	Default
Service	●	○
Analytics	●	○
Marketing	●	○
+ Special data	○	○

Token eb1bf072-989a-494f-a157-9c34f



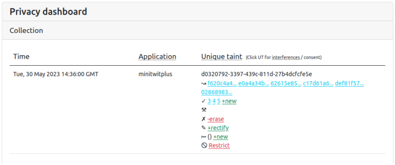
(b) The **Minitwit**$^{+}$ application

MiniTwit
my timeline | public timeline | sign out [user0]
My Timeline
What's on your mind user0?
Share

The Hague ... has the best conferences — MiniTwit Ad

user0 Hello world! — 2023-05-30 @ 12:36

MiniTwit — A GDPR-compliant Application

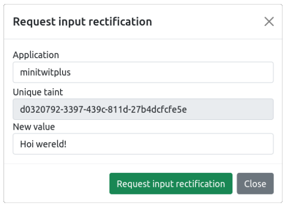


(c) The dashboard entry corresponding to the “Hello world” in Figure 6b

Privacy dashboard

Collection

Time	Application	Unique taint (Click UT for interferences / consent)
Tue, 30 May 2023 14:36:00 GMT	minitwitplus	d0320792-3397-439c-811d-27b4dcfcfe5e ↝ f620c4a4... e0a4a34b... 62615e85... c17d61a6... def81f57... 02868983... ✓ 3 4 5 +new ⚒ ✗ -erase ✎ +rectify ↦ () +new ⃠ Restrict



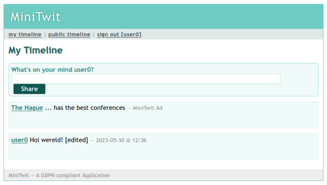
(d) Requesting rectification of “Hello world”

Request input rectification

Application: minitwitplus

Unique taint: d0320792-3397-439c-811d-27b4dcfcfe5e

New value: Hoi wereld!



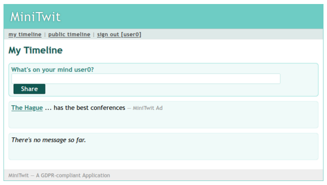
(e) After performing rectification

MiniTwit
my timeline | public timeline | sign out [user0]
My Timeline
What's on your mind user0?
Share

The Hague ... has the best conferences — MiniTwit Ad

user0 Hoi wereld! [edited] — 2023-05-30 @ 12:36

MiniTwit — A GDPR-compliant Application



(f) After revoking consent for “Hello world”

MiniTwit
my timeline | public timeline | sign out [user0]
My Timeline
What's on your mind user0?
Share

The Hague ... has the best conferences — MiniTwit Ad

There's no message so far.

MiniTwit — A GDPR-compliant Application

C Inaccuracies in Arfelt et al. [2]

1. Arfelt et al.'s specification lacks $\blacklozenge\,\texttt{Collect}(ds, ut, sp)$ conjuncts, allowing consent given by *any user* to justify data usage.
2. In the right to object, the condition “there has been an objection since legal grounds have been claimed” is $\exists \ldots \neg\,(\neg\texttt{DSObject}(\ldots)\ \mathsf{S}\ \texttt{LegalGround}(\ldots))$, allowing applications to reclaim the same legal grounds to override objections.