RuleMesh

ENGINEERED COMPLIANCE INFRASTRUCTURE

CASE STUDY · GDPR ARTICLE 22

Automated decision-making: the EU regulatory position and lessons from Uber's driver deactivations

The software behaved as designed.
What was missing from the specification?

PUBLICATION
2026

SOURCE OF TRUTH
RuleMesh GDPR rule graph
CELEX 32016R0679

CASE
Uber driver deactivations
Dutch DPA and CNIL

AUTOMATED DECISION-MAKING

Automated decision-making: the EU regulatory position and lessons from Uber's driver deactivations

What Uber's automated driver deactivations can teach product and engineering teams about translating regulation into requirements before a system is built.

RuleMesh · 27 August 2026 · 10 minute read

The difficult thing about compliance failures in complex organisations is that they rarely begin with a team deciding to ignore the law.

More often, a product team is solving a real operational problem. An engineering team is asked to make the solution reliable and scalable. Legal and compliance teams are working with a regulation written for a much wider set of circumstances. Each group may be doing reasonable work within its own field of view, while something important remains between them.

Uber may have had that kind of blind spot. We do not know its internal development process, and this case study does not speculate about intent. The public decision does, however, expose a gap that many large organisations will recognise: the law constrained how a digital decision was allowed to happen, but that constraint did not appear to have been expressed as product behaviour at the point where the system needed it.

The age of the rule makes the gap more striking. The GDPR has applied since 2018, and its protection against certain solely automated decisions has a direct predecessor in Article 15 of the EU's 1995 Data Protection Directive. By 2026, the core principle was more than 30 years old. This was not a new rule written in response to the current AI cycle.

An organisation can invest seriously in privacy, build a mature compliance programme and sincerely consider itself GDPR compliant, while a particular safeguard never becomes an executable product requirement. Uber may have considered itself compliant. We do not know. The

point is that access to the law does not guarantee that every obligation reaches the people and systems that must implement it.

We chose this case because it makes that difficult gap unusually clear, not to trade on Uber's difficulty. Compliance is no longer confined to policies, access controls or an audit folder. Regulation can determine whether a system may make a decision, what must happen before that decision becomes final, how a person can challenge it, and what evidence must remain afterwards. New approaches are needed to carry that intent into the work of building software. RuleMesh is our approach to that problem.

> "Our choice is not between 'regulation' and 'no regulation.' The code regulates."
>
> Lawrence Lessig, Code Is Law, Harvard Magazine, 2000

What happened

More than 170 drivers brought a collective complaint in France concerning the way Uber used automated systems to suspend or deactivate driver accounts. Because Uber's main EU establishment is in the Netherlands, the Dutch Data Protection Authority led the cross-border investigation with France's CNIL.

The Dutch DPA said that suspected fraud and low customer ratings could trigger temporary or permanent deactivation without human intervention in the decision itself. For a driver, the effect was not abstract: a blocked account meant no access to rides and no ability to earn through the platform.

On 21 August 2026, the authority announced a fine of €824.99 million. Uber has said it will appeal. The eventual legal process may continue, but the engineering lesson is already visible: the disputed act was executed by the product.

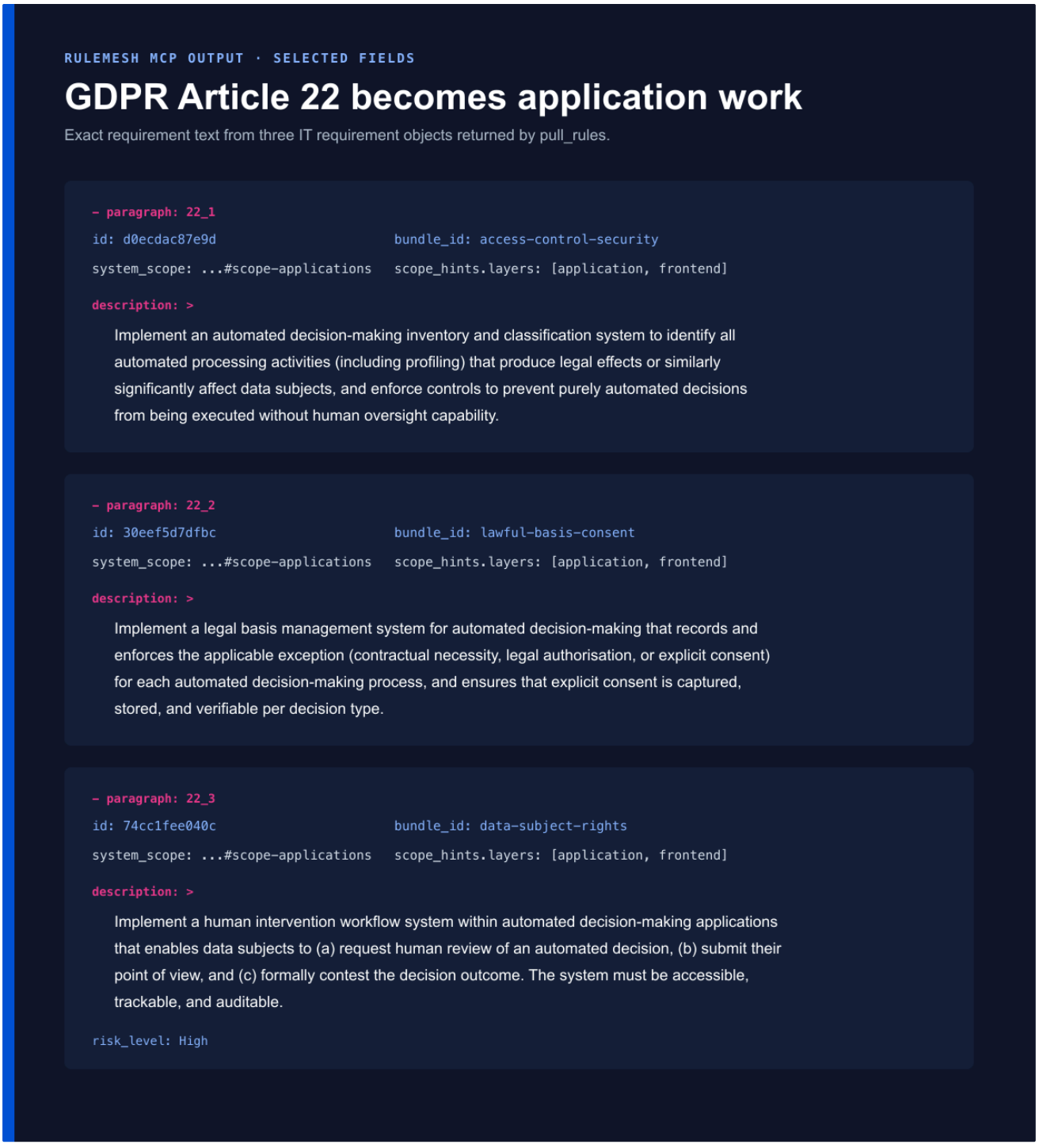
The behaviour at issue was a state transition, not a missing policy document.

Why this was a software-behaviour problem

A conventional compliance review might ask whether an algorithm policy existed, whether support staff had been trained, whether access was controlled, or whether system events were logged. Those questions matter. None of them determines whether the software can move a driver from `flagged` to `deactivated` without a meaningful human decision.

That distinction matters because security controls and product semantics do different work. Authentication can restrict who reviews a case. Logging can record what happened. Neither creates a right to request review, lets a driver submit their point of view, gives an authorised reviewer power to change the outcome, or stops an automated result from becoming final.

The relevant compliance requirement therefore belonged in the state model, service logic, interface, reviewer workflow, release criteria and tests. A policy could describe the safeguard; only the product could deliver it.



Selected fields from the RuleMesh MCP response, rendered without changing the requirement text. The legal provisions become application-level work.

The obligations behind the behaviour

GDPR Article 22(1) sets the boundary. A person has the right not to be subject to a decision based solely on automated processing, including profiling, when it produces legal effects or similarly significant effects. Losing access to platform work is the kind of consequence that brings the behaviour into focus.

GDPR Articles 22(2) and 22(3) deal with exceptions and safeguards. Sole automation is permitted only in narrow circumstances: contractual necessity, authorisation by law, or explicit consent. In contract and consent cases, the controller must at least enable human intervention, let the person express their point of view, and let them contest the decision. Those are not general aspirations; they describe capabilities and decision states.

GDPR Articles 13(2)(f), 14(2)(g) and 15(1)(h) create transparency and access obligations. People must receive meaningful information about the logic involved, the significance of the processing, and its envisaged consequences. That implies an inventory of automated decision systems and information that can actually be retrieved for a response.

GDPR Articles 35(1) and 35(3)(a) create an upstream design gate. A data protection impact assessment is required before high-risk processing begins. Systematic and extensive automated evaluation used for decisions with legal or similarly significant effects is an express trigger.

GDPR Article 25(1) places safeguards in the design process. Appropriate technical and organisational measures must be integrated when the means of processing are determined and while processing takes place, not reconstructed after release.

> The Dutch DPA decision directly concerns automated decision-making. GDPR Articles 25 and 35 are included here as upstream obligations represented in the RuleMesh data that would be available during design. We are not presenting them as additional findings made in the fine.

Where a generic checklist runs out

A checklist can remind a team that GDPR Article 22 exists. It cannot, by itself, express the application behaviour contained in the IT requirements shown above: inventory the decision system, record the applicable exception, prevent finalisation without human-review capability, and provide a contestation workflow.

This is also where general AI knowledge reaches its limit. A coding agent may know that the GDPR contains rules about automated decisions. Asked to “make the feature GDPR compliant,” it can produce a plausible answer. But model memory is not an authoritative source for the organisation’s scope decision, accepted interpretation, implementation requirement or evidence standard. A fluent answer is not the same thing as a traceable specification.

For this case study, we start with the behaviour described in the public decision. We then use the RuleMesh MCP to retrieve the structured IT requirements linked to the relevant GDPR provisions. This is not a RuleMesh assessment of Uber, and no bespoke case profile has been created. The

material shown here is regulatory data returned by the product: source references, IT requirement text, system scope, completion checks, evidence expectations and control mappings.

What the RuleMesh data gives the team

The value is not the article number. It is the operational data attached to it: a stable requirement ID, application scope, the IT requirement, completion criteria, expected evidence and control mappings where a generic control genuinely helps. A person can read it in the RuleMesh product. A coding agent can retrieve the same object through MCP.

1. Detect the regulated decision path

`GDPR ARTICLE 22(1) · REQUIREMENT d0ecdac87e9d`

The IT requirement returned for GDPR Article 22(1) calls for an inventory and classification system for automated processing that produces legal or similarly significant effects. It also calls for controls that prevent purely automated decisions from being executed without human-oversight capability.

- **Done means:** the decision is inventoried, classified by impact, and unable to become final without the required review capability.
- **Evidence includes:** decision-pipeline architecture, impact assessments, the automated-decision register and human-review workflow documentation.

2. Make the permitted exception executable

`GDPR ARTICLE 22(2) · REQUIREMENT 30eef5d7dfbc`

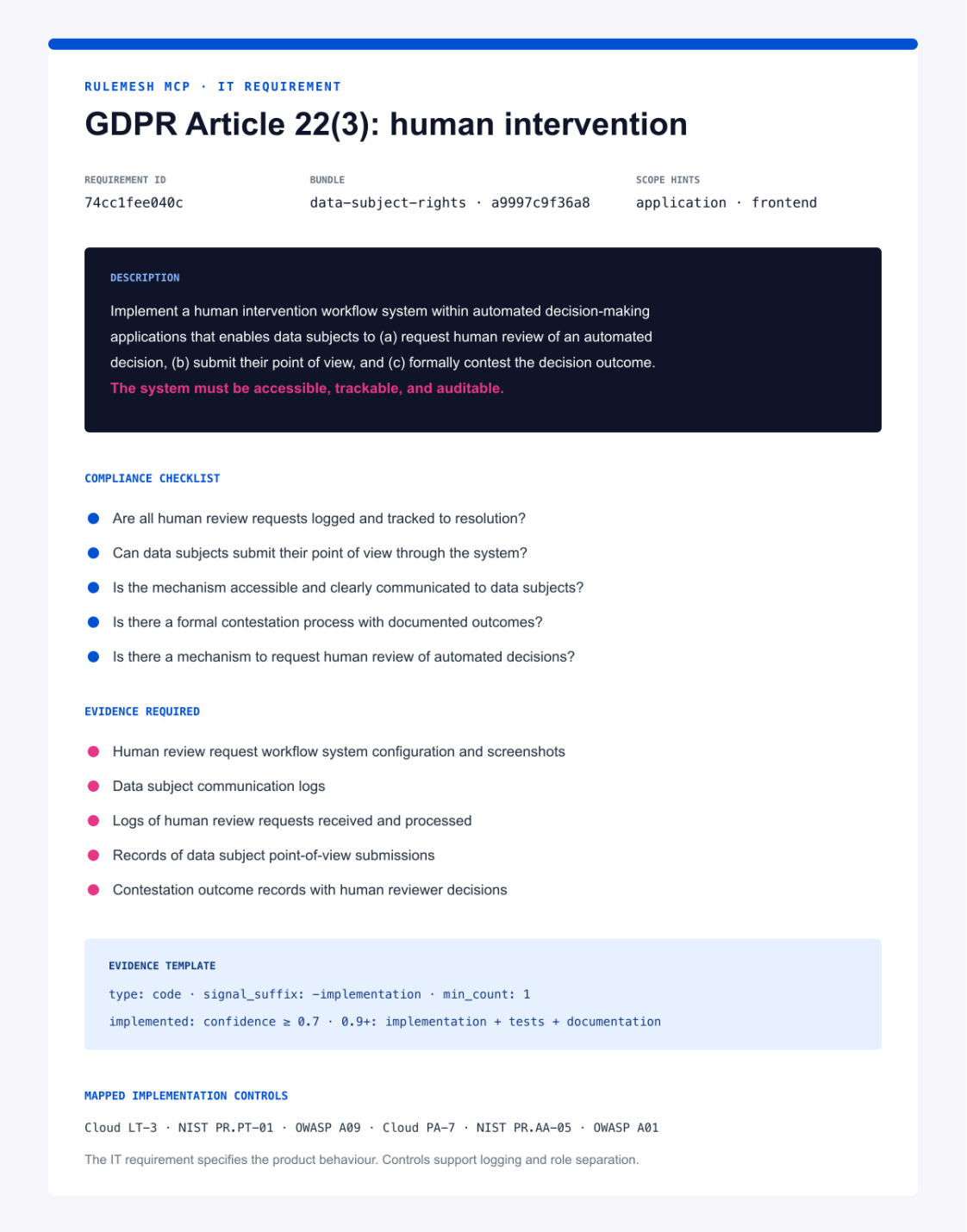
The IT requirement returned for GDPR Article 22(2) calls for a legal-basis management system that records and enforces the applicable exception for each automated decision process: contractual necessity, legal authorisation or explicit consent. The basis cannot remain an unsupported sentence in a policy.

- **Done means:** the basis is recorded per decision process; necessity is assessed rather than asserted; law-based safeguards or explicit consent can be verified.
- **Evidence includes:** necessity assessments, legal-authorisation documentation, a legal-basis register and granular, timestamped consent records where relevant.

3. Build the human-intervention workflow

`GDPR ARTICLE 22(3) · REQUIREMENT 74cc1fee040c`

The IT requirement returned for GDPR Article 22(3) calls for an accessible, trackable and auditable workflow through which a person can request human review, submit their point of view and formally contest the outcome. The requirement object also carries the checklist, expected evidence, evidence-submission template and mapped implementation controls.



Rendered from RuleMesh IT requirement 74cc1fee040c in the Data Subject Rights Operations bundle. The wording and fields come from the MCP response.

For a technical reader, this is the DSL in practical terms. The requirement is not only presentation copy in a user interface. Through MCP, a coding agent receives a structured, machine-readable object. The excerpt below uses the actual call, field names and values returned for requirement

74cc1fee040c. Only unrelated response fields are omitted.

RULEMESH MCP · DSL VIEW

GDPR Article 22(3), as a coding agent receives it

An actual pull_rules call and selected fields from the returned requirement object.

MCP CALL

```
pull_rules({ "bundle_id": "data-subject-rights", "compact": false })
```

RETURNED REQUIREMENT · SELECTED FIELDS


```
{
  "id":"74cc1fee040c",
  "article":"22",
  "paragraph":"22_3",
  "risk_level":"High",
  "system_scope":"http://rulemesh.com/vocab/regulation#scope-applications",
  "scope_hints": {"layers": ["application","frontend"] },
  "compliance_checklist": [
    "Are all human review requests logged and tracked to resolution?",
    "Can data subjects submit their point of view through the system?",
    "Is the mechanism accessible and clearly communicated to data subjects?",
    "Is there a formal contestation process with documented outcomes?",
    "Is there a mechanism for data subjects to request human review of automated decisions?"
  ],
  "evidence_template": {
    "bundle_id":"a9997c9f36a8",
    "requirement_id":"74cc1fee040c",
    "evidence_submissions": [{
      "type":"code","signal_suffix":"-implementation"
    }],
    "min_count":1,
    "confidence_guide": {
      "0.9+":"Implementation found with tests and documentation",
      "0.7-0.9":"Implementation found in source code"
    }
  }
}
```


RuleMesh MCP output for GDPR Article 22(3), selected for readability. The field names and values shown are unchanged.

4. Make logic and consequence retrievable

GDPR ARTICLE 15(1)(h) · REQUIREMENT 3852ff35fd79

The IT requirement returned for GDPR Article 15(1)(h) calls for a subject-access process that can retrieve the logic, significance and consequences of relevant automated decision systems and track an electronic request against the statutory response window.

- **Done means:** the information is documented and retrievable across relevant systems and responses are tracked against the one-month deadline.
- **Evidence includes:** request-channel configuration, sample reports, processing inventory, request tracking and response-timeline logs.

5. Turn the DPIA into a release gate

`GDPR ARTICLE 35(1) + 35(3)(a) · REQUIREMENT 107b148c25c6`

The IT requirement returned for GDPR Articles 35(1) and 35(3)(a) calls for new and changed processing to be screened before it starts. It also calls for systematic automated evaluation with significant effects to be flagged, and for commencement to be prevented until the required assessment and approval are recorded.

- **Done means:** screening occurs before commencement, the correct trigger fires, and high-risk processing cannot launch without completion and approval.
- **Evidence includes:** screening decisions, completed DPIAs, pre-processing approvals and a register showing rationale, outcome and status.

What this changes for a product team and its coding agent

Imagine that Codex is asked to build the deactivation feature. With a generic prompt, it can reasonably create a fraud-score event, a threshold rule, a deactivation endpoint, a notification and an operational log. The implementation may be clean. The tests may pass. The wrong behaviour may reach production faster.

With RuleMesh available through MCP or project context, the same agent receives requirement `74cc1fee040c` and its evidence contract. It can inspect the existing repository and turn that requirement into work that a human team can review:

1. insert `pending_human_review` before `deactivated` and block automatic finalisation;
2. add review-request, point-of-view and contestation interfaces and APIs;
3. route the case to an authorised reviewer with power to change the outcome;
4. record the request, communication, reviewer decision and resolution time;
5. map tests to each RuleMesh completion criterion; and
6. return implementation, test and documentation signals against the same requirement ID.

That is a different use of AI. The agent is not being asked to invent the regulatory interpretation. It is being asked to implement a structured requirement whose source, scope and evidence expectation are already visible. Humans still confirm the high-stakes legal conclusion and review the code; RuleMesh makes sure they are reviewing the right question.

Lessons from the case

1. **Compliance can be a property of system behaviour.** The decisive question may be whether a state transition is allowed to occur, not whether a policy exists.

2. **People cannot design for obligations they never receive.** Legal context has to enter product discovery, acceptance criteria and technical planning early enough to change the design.

3. **Generic checklists do not express product semantics.** They can support governance, but they do not create a review journey or block an unlawful automated outcome.

4. **Atomic requirements make regulation buildable.** Stable IDs, scope, completion criteria and evidence turn a broad legal rule into work that engineers and agents can implement and test.

5. **Evidence should emerge from delivery.** Architecture, workflow configuration, tests and outcome logs should be linked to the requirement while the system is built.

Why RuleMesh

This case does not show that product teams need more legal text. They already have access to the GDPR. It shows that they need infrastructure between legal intent and system delivery.

RuleMesh structures source law as atomic IT requirements with completion checks, system scope, evidence expectations and control mappings. Product, engineering, security and compliance teams, along with the coding agents working beside them, can use the same trace before behaviour ships.

That does not remove judgement, and it does not guarantee that reasonable people will never disagree about scope. The organisation still makes its legal and risk decisions. RuleMesh keeps the source, requirement, implementation criteria and evidence expectation connected, giving the people building the system something precise enough to work with. In complex regulatory environments, that missing layer is the difference between knowing that a rule exists and knowing what the product must do.

Regulation increasingly defines how digital systems are allowed to behave, but the teams designing that behaviour often do not have the regulatory context they need.

Case sources: Dutch DPA and CNIL.

Legal and requirement trace: RuleMesh MCP, GDPR knowledge graph, CELEX 32016R0679. Retrieved 27 August 2026. Requirement IDs refer to the derived RuleMesh IT requirements described in the article.

Historical context: EU Data Protection Directive 95/46/EC, Article 15, and EUR-Lex guidance on GDPR application from 25 May 2018.

Back to top