The Case for XACML-Driven ABAC in Zero Trust Architectures

Overcoming Policy Complexity with GUI-Based No-Code Administration

Ross Brenner

April 2025

April 2025

Contents

Executive Summary

Government agencies, enterprise CISOs, and military decision-makers face a critical challenge: how to implement fine-grained Zero Trust Architecture (ZTA) without creating a brittle, unmanageable policy environment. ABAC has emerged as a cornerstone of ZTA, providing dynamic, context-rich access decisions.

Among Attribute-Based Access Control (ABAC) standards, Extensible Access Control Markup Language (XACML) stands out as the most mature and interoperable. Backed by NIST guidance, embraced by NATO for secure coalition information sharing, and integrated into large-scale identity frameworks (from SAML to OAuth), XACML offers a proven model for policy interoperability across diverse systems.

However, XACML's complexity in authoring has historically limited its broader adoption. Writing verbose XML policies, even for simple rules, is daunting for non-developers and increases the likelihood of syntax errors. These errors introduce unnecessary friction into the policy lifecycle and, in the context of a critical security application, can create operational risks—either through misconfiguration or delays in policy rollout.

After investigating XACML-driven ABAC, this white paper argues that GUI-based policy creation and administration removes the complexity barrier. This makes ABAC policy management more accessible to security architects, administrators and auditors alike.

By abstracting the complexity of XACML and hiding XML intricacies behind intuitive interfaces, tools like [ViewDS Access Sentinel]() enable Zero Trust policies to be defined, tested and enforced with agility—without compromising rigour or expressiveness.

Key takeaways:

- ABAC is foundational to ZTA; attempting Zero Trust without ABAC leads to static, brittle policies that cannot adapt to context.
- XACML is the leading ABAC standard, endorsed by standards bodies and defence alliances for its expressiveness and interoperability.
- The main hurdle with XACML is its XML-based complexity, *not* its capability. A simple policy can span dozens of lines of XML, or multiple definitions in Rego, confusing those without developer training.
- Modern tools like ViewDS Access Sentinel provide a no-code environment to write ABAC policies. Visual policy construction with drag-and-drop selection allows policy admins to focus on logic, not syntax.

Update: XACML Next Generation

Since this white paper was written, significant progress has been made towards the release of XACML's next major version, informally referred to as 'XACML Next Gen' or XACML 4.0.

This upcoming release introduces significant improvements, such as:

- *Syntax-agnostic policy representations (supporting JSON and YAML in addition to XML),*
- *Simplified policy structures (combining Policy and PolicySet constructs), and*
- *The addition of global variables and composite functions.*

Collectively, these improvements aim to reduce policy verbosity and complexity, aligning closely with the emphasis on lowering adoption barriers for non-technical policy administrators.

While these improvements make XACML policies more intuitive and manageable, the need for comprehensive GUI-based policy builders, such as ViewDS Access Sentinel, remains critical. Such tools continue to provide essential abstraction, ensuring broad accessibility and ease-of-use in large-scale deployments.

It is noteworthy that ViewDS's CTO, Dr. Steven Legg, is an active member of the Organization for the Advancement of Structured Information Standards (OASIS) XACML Technical Committee and has made substantial contributions to the development of XACML Next Gen.

ABAC: The Cornerstone of Zero Trust Architecture

ZTA shifts the security paradigm from perimeter-based defences to continuous, attribute-driven access decisions. As *NIST SP 800-207* *(Zero Trust Architecture)* emphasises, "Zero Trust is built upon attribute-based access control (ABAC)", which means that each access request should be dynamically evaluated based on attributes of the user, resource, environment and request context.

Why ABAC for Zero Trust?

Traditional models like RBAC assign static roles to users (for instance, Alice is an "HR Manager"), which then grant permissions. This static nature struggles to accommodate the dynamic, contextual decisions Zero Trust requires.

Static vs Dynamic

RBAC roles often map to organisational positions. They do not easily account for contextual factors such as time of day, device posture, threat level. A role like "Manager" might, for instance, grant broad access to sensitive data regardless of the individual's current risk posture—a brittle and overly permissive approach that goes against the principles of Zero Trust.

ABAC, on the other hand, allows much finer control. A policy might state that access is only permitted if the user has the "Manager" role *and* their device is compliant, *and* they've successfully completed multi-factor authentication within the last 60 minutes. Such dynamic, attribute-driven policies support the core Zero Trust principle: "Never Trust, Always Verify."

Combining Factors

RBAC alone cannot handle multi-factor decisions well. ABAC can combine arbitrary attributes—user role, clearance level, data sensitivity label, network location, etc.—to make nuanced decisions. For example, to access a specific resource, ABAC can enforce that a user must meet the following criteria:

- *role is Manager*,
- *clearance is NV1*, and
- *have completed MFA authentication in the last 60 minutes*.

In many enterprise and government environments, these attributes already exist in authoritative systems such as LDAP directories or personnel databases. ABAC enables organisations to leverage these sources directly, streamlining integration between identity management and policy enforcement—a key benefit when implementing Zero Trust at scale.

Least Privilege & Adaptability

Zero Trust mandates least privilege access. ABAC policies can be finely scoped to contextual conditions, enabling just-in-time access when supported by dynamic inputs and policy orchestration.

RBAC roles, by contrast, often grant broad permissions that risk over-entitlement. When requirements shift, RBAC systems typically require manual role reconfiguration, while ABAC policies can be updated centrally to reflect new rules in real time.

Industry and Government Endorsement

Recognising these benefits, NIST provides a formal definition of ABAC in *SP 800-162,* highlighting it as a flexible model to meet complex access needs. The U.S. Department of Defense's *Zero Trust Reference Architecture* further reinforces the importance of dynamic, attribute-based access decisions as foundational to Zero Trust principles.

Within the Australian Defence Force, Data-Centric Security initiatives also recognise the need for ABAC controls to protect mission-critical, encryted data—reinforcing ABAC's essential role in secure coalition interoperability.

Why XACML is the Premier Standard for ABAC

XACML, standardised by OASIS, has been available since 2003 and serves as both a policy language and a reference architecture for ABAC. It defines how to express "who can do what under which conditions" in a structured policy format, and how to evaluate policies against access requests.

Key reasons why XACML is widely considered the most suitable ABAC standard include the following:

- **Maturity & Adoption:** With over two decades of development, XACML has matured through real-world use. It underpins many enterprise authorisation solutions and frameworks. NIST's comparative study of ABAC standards (*SP 800-178*) frames XACML as a baseline for modern authorisation, noting it "is the basis of many modern authorisation solutions".
- **Interoperability:** XACML's strength lies in its standardisation. Policies are expressed in a vendor-neutral XML (with XACML 4.0 also supporting JSON and YAML) structure, meaning different implementations (for example, open-source PDPs, commercial policy servers) can share and enforce the same policies. This is vital for government and military contexts where multi-vendor, multi-domain interoperability is required. NATO's federated missions and coalition networks benefit from a common policy language to share access rules across nations.
- **Expressive Power:** XACML supports rich policy constructs: rules, policies, policy sets, with combining algorithms (for example, first applicable, permit-overrides) to render decisions. It covers attributes for subjects, resources, actions and environments, enabling complex logic (time-of-day restrictions, numeric comparisons, set-based operations, hierarchical resources, etc.). It also supports obligations and advice to trigger audit logs, masking or other actions as part of the decision.
- **Backed by Standards Bodies:** NIST's ABAC guide (*SP 800-162*) uses XACML for many examples and references, implicitly endorsing it as a canonical approach for policy definition. XACML is often referenced in government ICAM strategies as a way to externalise and standardise access policies. Large identity frameworks like SAML and OAuth can carry XACML policy or attribute information, and some IdPs allow XACML-based authorisation claims to be evaluated post-authentication.

Comparison to Alternatives

The primary alternative ABAC model in NIST's study is Next Generation Access Control (NGAC), a graph-based system emphasising relationships. While NGAC is promising, its adoption remains largely confined to research environments and specialised applications. More recently, Open Policy Agent (OPA), utilising its Rego policy language, has seen rapid adoption, particularly within cloud-native and DevOps environments. OPA/Rego's popularity stems from its simplicity and flexible, code-centric approach.

A detailed comparison between XACML and OPA/Rego is beyond the scope of this white paper and would obscure the primary challenge outlined above. A comprehensive analysis, including detailed use cases and code examples comparing XACML and OPA/Rego, will be provided in a separate, dedicated white paper. However, in brief:

- **XACML** delivers a comprehensive and standardised ABAC framework including a clearly defined architecture (PAP, PDP, PEP). Its strength lies in extensive interoperability and alignment with established identity standards.
- **OPA/Rego** provides a lightweight, code-oriented policy engine well-suited for cloud-native microservices, APIs and infrastructure-as-code deployments. However, it does not yet offer a universally agreed-upon schema for attributes, which can introduce interoperability challenges in heterogeneous, multi-vendor environments.

In fact, Styra, the company behind OPA, acknowledges XACML's foundational role, stating clearly that "the XACML architecture… still represents the model for how OPA handles integrations, just in a more modern, distributed way."

Why Policy Complexity Slows ABAC Adoption

If XACML is so powerful, why isn't every organisation using it? The answer lies in usability and expertise. Authoring XACML by hand means writing XML with nested elements, intricate identifiers and functions—a format not friendly to humans. For example, consider a simple policy requirement: *Only allow SECRET emails to go to recipients with SECRET clearance.*

In XACML XML, this might be represented as:

```
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" PolicyId="urn:uuid:6628ae24-6fdb-
4aa9-bdce-a443ba1ceb20" RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-
algorithm:deny-overrides">
   <Rule RuleId="deny-secret-without-clearance" Effect="Deny">
```

```
      <Condition>
         <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">
            <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:string-contains">
               <AttributeValue DataType="[http://www.w3.org/2001/XMLSchema#string]
                (http://www.w3.org/2001/XMLSchema#string)">SECRET</AttributeValue>
               <Attribute DesignatorAttributeId="[http://viewds.com/xacml/action/email/subject]
                (http://viewds.com/xacml/action/email/subject)"
                Category="urn:oasis:names:tc:xacml:3.0:attribute-category:action"
                DataType="[http://www.w3.org/2001/XMLSchema#string]
                (http://www.w3.org/2001/XMLSchema#string)" MustBePresent="true" />
            </Apply>
            <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:not">
               <Apply FunctionId="urn:oasis:names:tc:xacml:3.0:function:string-equal-ignore-case">
                  <AttributeDesignator AttributeId="[http://viewds.com/xacml/resource/clearance]
                   (http://viewds.com/xacml/resource/clearance)"
                   Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
                   DataType="[http://www.w3.org/2001/XMLSchema#string]
                   (http://www.w3.org/2001/XMLSchema#string)" MustBePresent="true" />
                  <AttributeValue DataType="[http://www.w3.org/2001/XMLSchema#string]
                   (http://www.w3.org/2001/XMLSchema#string)">SECRET</AttributeValue>
               </Apply>
            </Apply>
         </Apply>
      </Condition>
   </Rule>
</Policy>
```

This policy defines a rule that denies access when an email's subject contains the word "SECRET" and the recipient's clearance is not "SECRET". Even this simple policy requires familiarity with XACML's XML schema, attribute categories, functions and combining algorithms. Small mistakes—such as typos in URNs, incorrect attribute references or missing MustBePresent flags—can lead to unintended behaviour or security gaps.

In OPA's Rego language, the equivalent logic might be expressed as:

```
package email.policy
default allow = false
deny["recipient must have SECRET clearance for SECRET emails"] {
  contains(lower(input.action.subject), "secret")
  lower(input.resource.clearance) != "secret"
}
allow {
  not deny[_]
  not contains(lower(input.action.subject), "secret")
} else {
  contains(lower(input.action.subject), "secret")
  lower(input.resource.clearance) == "secret"
}
```

This Rego policy is certainly more concise than its XACML counterpart. However, it introduces its own challenges: it assumes a specific input data model (for example, input.action.subject and input.resource.clearance) that must be clearly defined and consistently followed.

It also requires familiarity with a programming-style syntax, which can be a barrier for policy authors and auditors without a development background. As policies grow more complex, Rego can become harder for non-developers to write, interpret or audit compared to more declarative formats.

The Missing Link: How to Overcome XACML's Complexity

Both XACML XML and Rego assume policy authors are comfortable with code or markup. For many government and enterprise policy administrators and auditors—often analysts or architects with deep security expertise but limited programming experience—this presents a significant barrier. During a recent industry consultation, in which ViewDS participated, a senior technical expert at the Australian Signals Directorate echoed a widely held concern:

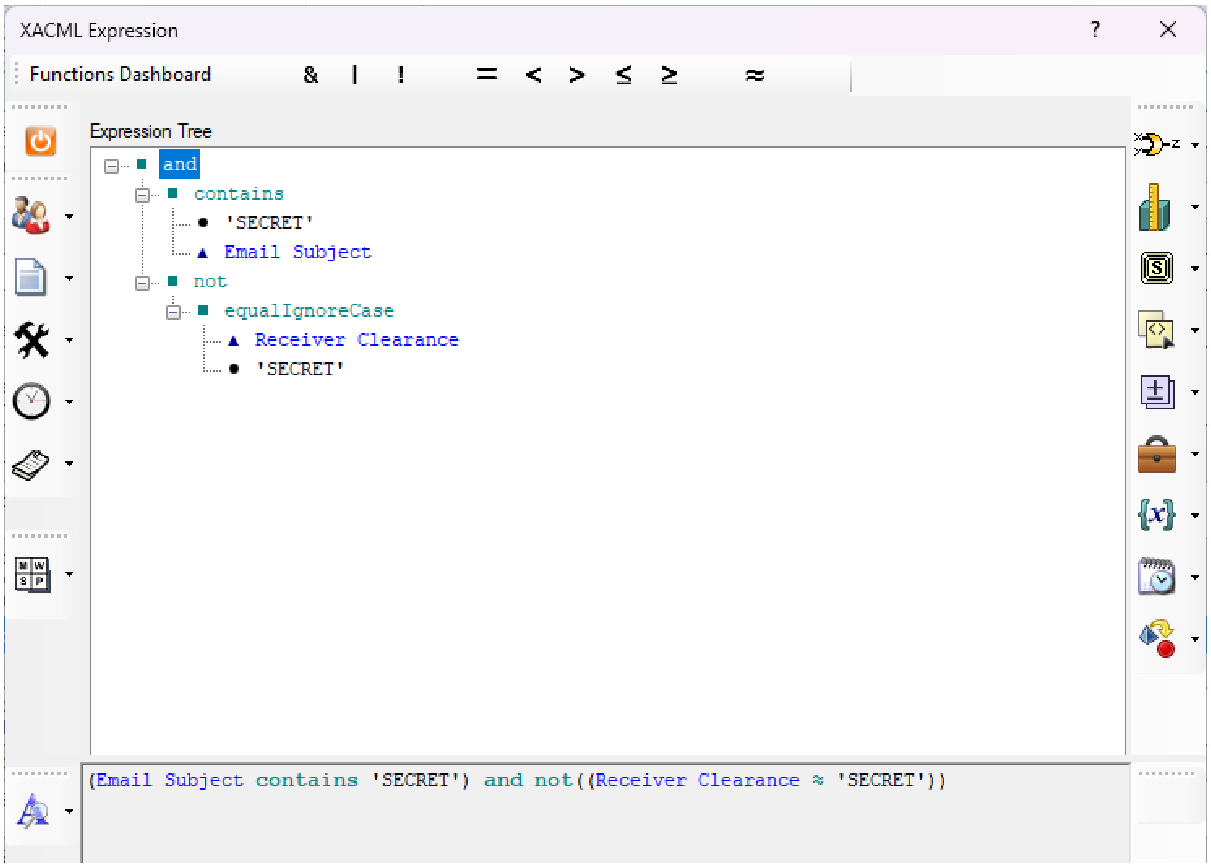
> …the complexity of authoring policies remains a key obstacle to adoption. Organisations frequently find themselves constrained by the perception that only developers have the skills to manage and implement authorisation policies. This results in bottlenecks, silos and ultimately slows the adoption of ABAC and Zero Trust architectures.

ViewDS directly addresses this challenge by empowering non-developer policy administrators through intuitive, GUI-driven tools that simplify policy creation and management.

A well-designed GUI for ABAC policy creation can hide the complexity of XACML without sacrificing its power. The idea is to let users define policies through intuitive actions: selecting attributes from drag-and-drop menus and dropdowns, drawing relationships, using forms to specify conditions. This is all while the tool generates XACML under the hood.

ViewDS Access Sentinel provides a PAP GUI that exemplifies this approach (see Figure 1 below). In Figure 1, an administrator is visually constructing an XACML condition that enforces clearance-based email protection. The expression tree in the centre panel shows a logical and operation composed of two key checks:

- The Email Subject contains "SECRET"
- The Receiver Clearance is not equal to "SECRET", achieved using a not block combined with a case-insensitive string comparison function, equalIgnoreCase. This rule implements the logic: "If the subject contains 'SECRET' and the recipient does not have SECRET clearance, deny access." The bottom panel of the GUI displays a natural-language summary of the rule being built: (Email Subject contains 'SECRET') and not((Receiver Clearance ≈ 'SECRET'))



```
and
  contains
    ● 'SECRET'
    ▲ Email Subject
  not
    equalIgnoreCase
      ▲ Receiver Clearance
      ● 'SECRET'
```

```
(Email Subject contains 'SECRET') and not((Receiver Clearance ≈ 'SECRET'))
```

Figure 1: Visual Policy Construction Using ViewDS Access Sentinel's GUI

In the above figure, note how the ViewDS Access Sentinel's GUI simplifies XACML policy creation by abstracting low-level syntax:

- **Subject and Resource Attribute Selection:** Administrators select from a human-readable list (for example, "Email Subject"), eliminating the need to recall complex URNs like http://viewds.com/xacml/action/email/subject.
- **Function Handling:** The GUI offers context-aware function options (for example, contains, equalIgnoreCase) based on attribute types, while automatically preventing invalid combinations.
- **Logical Composition:** Logical blocks (AND, OR, NOT) are constructed visually through drag-and-drop, ensuring correct nesting and adherence to XACML schema.
- **Policy Semantics:** Dropdowns and guided fields manage configuration details like combining algorithms (for example, "deny-overrides"), reducing error rates and training time.

The user interface allows administrators to create policies without writing code, while generating valid XACML policy files in the background. This approach directly addresses the barrier cited earlier: policy administrators can focus on the intent of the policy, not the syntax.

To further illustrate the value of this abstraction, consider the more complex policy described below. This rule governs access to documents under *STANAG 4774*, and expresses logic such as:

- **If** the action is ‘browse-document’ and the policy identifier is ‘NATO’ or ‘COSMIC’,
- **then** access may only be granted based on a combination of classification, clearance, succession timing and national release restrictions.

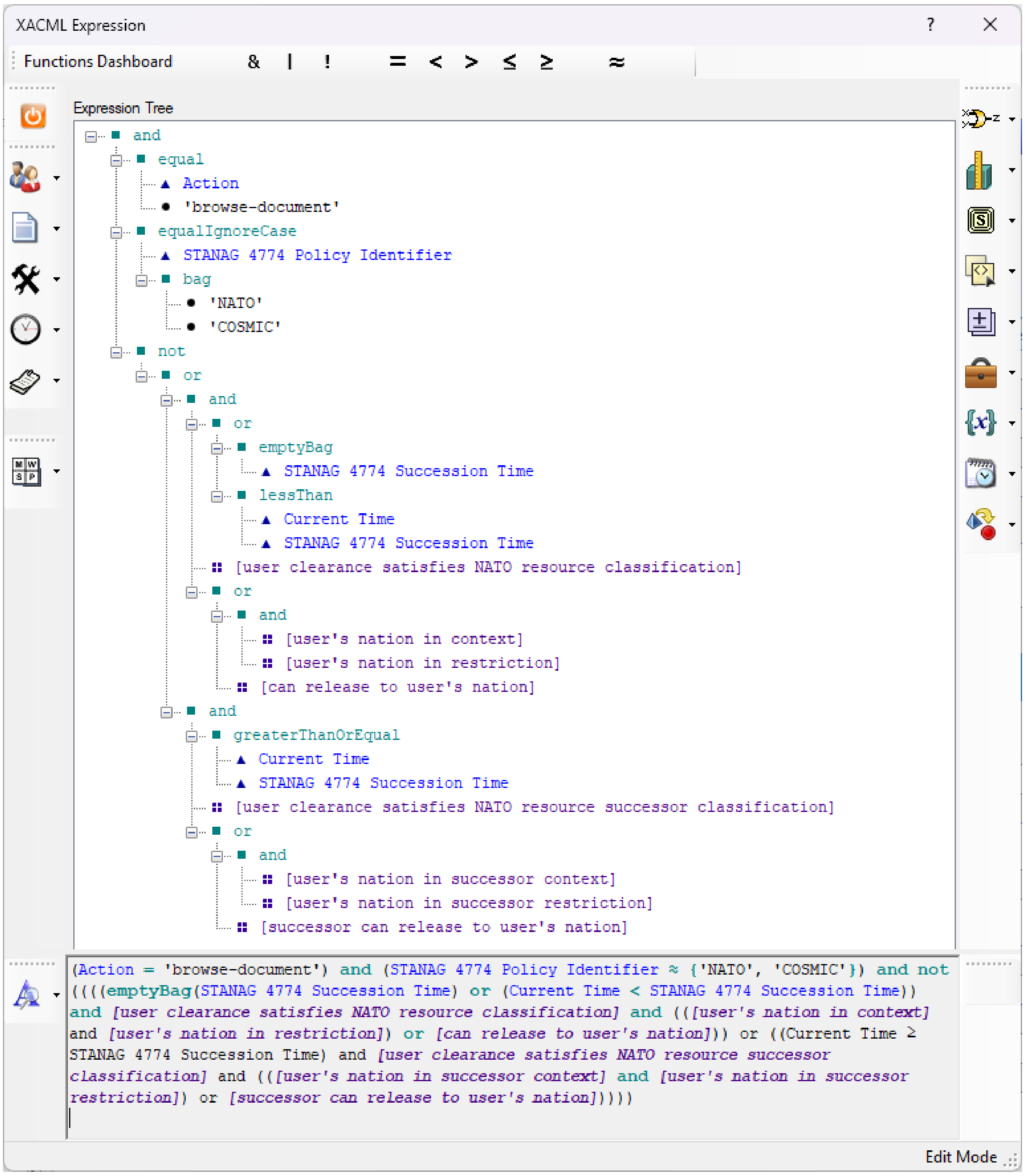
Despite being conceptually straightforward, this policy requires 413 lines of XACML XML, which is impractically complex for most policy authors. The equivalent policy in OPA/Rego still spans 30 lines of code, requiring a clear understanding of syntax, function composition and data models.

Figure 2 (below) shows this complex rule constructed visually in the Access Sentinel GUI, without writing a single line of code.

In Figure 2, a policy administrator is using the Access Sentinel GUI to construct a complex XACML rule aligned with STANAG 4774 access requirements. The rule enforces multiple conditions that must be met before access to a document is granted.

The conditions include:

- The action is ‘browse-document’
- The policy identifier is either ‘NATO’ or ‘COSMIC’
- The user’s clearance level meets or exceeds the document classification
- Succession timing constraints are met
- National release restrictions are satisfied



```
■ and
    ■ equal
        ▲ Action
        ● 'browse-document'
    ■ equalIgnoreCase
        ▲ STANAG 4774 Policy Identifier
        ■ bag
            ● 'NATO'
            ● 'COSMIC'
    ■ not
        ■ or
            ■ and
                ■ or
                    ■ emptyBag
                        ▲ STANAG 4774 Succession Time
                    ■ lessThan
                        ▲ Current Time
                        ▲ STANAG 4774 Succession Time
                ■ [user clearance satisfies NATO resource classification]
                ■ or
                    ■ and
                        ■ [user's nation in context]
                        ■ [user's nation in restriction]
                    ■ [can release to user's nation]
            ■ and
                ■ greaterThanOrEqual
                    ▲ Current Time
                    ▲ STANAG 4774 Succession Time
                ■ [user clearance satisfies NATO resource successor classification]
                ■ or
                    ■ and
                        ■ [user's nation in successor context]
                        ■ [user's nation in successor restriction]
                    ■ [successor can release to user's nation]
```

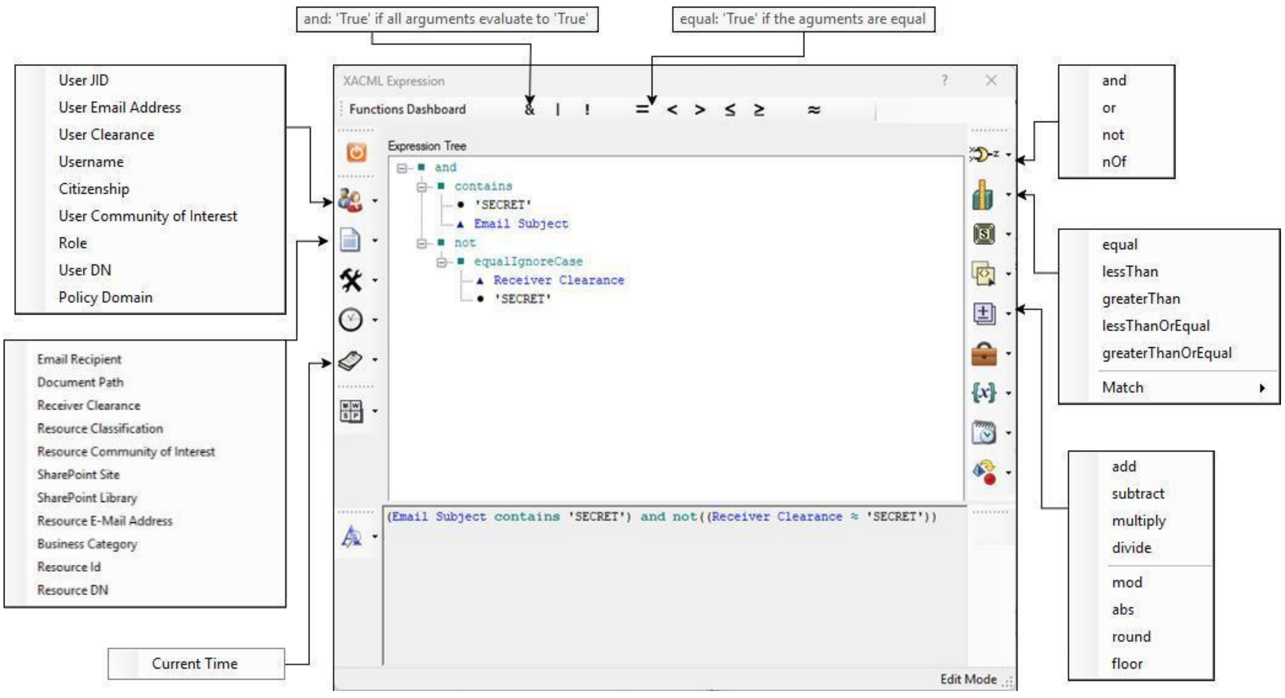
```
(Action = 'browse-document') and (STANAG 4774 Policy Identifier ≈ {'NATO', 'COSMIC'}) and not
((((emptyBag(STANAG 4774 Succession Time) or (Current Time < STANAG 4774 Succession Time))
and [user clearance satisfies NATO resource classification] and (([user's nation in context]
and [user's nation in restriction]) or [can release to user's nation])) or ((Current Time ≥
STANAG 4774 Succession Time) and [user clearance satisfies NATO resource successor
classification] and (([user's nation in successor context] and [user's nation in successor
restriction]) or [successor can release to user's nation])))
```

Figure 2: Visual Construction of a STANAG 4774 Access Control Rule

Empowering Policy Admins with Access Sentinel's No-Code GUI

To truly realise the potential of ABAC and Zero Trust, policy creation and management must be accessible beyond the developer silo. Access Sentinel's policy builder delivers this democratisation.

The GUI is built for policy administrators, auditors, identity architects and security officers who understand security requirements, but want to avoid the complexity and overhead of hand-coding.



```
and
  contains
    'SECRET'
    Email Subject
  not
    equalIgnoreCase
      Receiver Clearance
      'SECRET'
```

```
(Email Subject contains 'SECRET') and not((Receiver Clearance ≈ 'SECRET'))
```

Figure 3: Comprehensive Attribute and Function Selection in ViewDS Access Sentinel

In Figure 3, the Access Sentinel GUI is shown during the construction of a complex XACML policy condition. The screenshot demonstrates how administrators can build rules by selecting from a structured library of attributes and functions, without writing a single line of code.

Key features include:

- **Attribute Panels (left):** Users can select from clearly grouped attributes, which eliminates the need to memorise complex URNs or attribute schemas.
  - Subject Attributes (for example, User Clearance)
  - Resource Attributes (for example, Resource Classification)
  - Environment Attributes (for example, Current Time)

- **Expression Tree (centre):** The condition logic is visually composed using Boolean operators such as and, or and not. Nested conditions are expressed through expandable, editable blocks, ensuring correct XACML semantics without the need for XML expertise.
- **Function Library (top and right):** Functions are categorised and context-aware. Examples include:
    - Boolean Functions: and, or, not, nOf
    - Relational Functions: equal, lessThan, greaterThanOrEqual, match
    - Arithmetic Functions: add, subtract, multiply, divide

This layout enables fast, accurate and error-resistant policy development using intuitive domain language. The resulting outputs are XACML-compliant and ready for operational deployment—removing a major barrier to ABAC adoption in Zero Trust environments.

Conclusion

Zero Trust success hinges on ABAC, and by extension on having an effective way to implement ABAC policies enterprise-wide. XACML, as the most established ABAC standard, provides the interoperability and expressiveness needed for the job. This is evidenced by endorsements from NIST and adoption in frameworks used by NATO and large governments. But to unlock XACML's full potential beyond niche use, we must conquer its complexity.

Access Sentinel GUI-based Policy Administration Point is a game-changer. It transforms ABAC policy authoring from a developer-only task to an inclusive process where architects and analysts can actively participate.

This democratisation also delivers more robust policies. They are *richer* by capturing nuanced business rules, *more accurate* through validation, and *easier to maintain* with version control and clarity. The result: organisations can enforce Zero Trust principles with fine-grained control *without creating a policy management nightmare*.

Government agencies and military decision-makers can be confident that *only the right people, under the right conditions, access the right resources—with clear* policy definitions and audit logs to back it up. Plus, access rights can be adjusted in real time to match operational tempo.

When planning Zero Trust or ICAM initiatives, consider the following recommendations:

1. **Adopt ABAC as a core authorisation model**, mapping out key subject, resource, action and environment attributes for your organisation's needs (refer to *NIST SP 800-162* for guidance on planning ABAC deployments).
2. **Use XACML as the policy language** to ensure any solution aligns with industry standards and can integrate with future systems. Verify that vendors support importing and exporting XACML, even if they have proprietary extensions.
3. **Invest in a GUI-driven ABAC management tool**. Look for features like attribute management, visual rule building, integrated testing and delegation support (the capabilities described in this paper). This will greatly reduce training needs and policy errors.
4. **Start with pilot projects**. Identify a segment of your environment (for example, one data domain or application) and implement ABAC controls via XACML and a GUI PAP. Use lessons learned to refine your attribute schema and rollout approach.
5. **Engage stakeholders**. Involve compliance, mission owners and IT early. Show them visual policies and incorporate their feedback. This builds trust in the system and ensures the policies reflect real-world requirements.

By following these steps, agencies can avoid "rigid, brittle" access enforcement and instead achieve an agile, resilient Zero Trust posture—where policy is *centrally defined, easily managed and ubiquitously enforced*. The technology and tools to do this are available now, blending the best of standards-based security with admin-oriented design.

About ViewDS Identity Solutions

ViewDS technology secures some of the world's most sensitive environments. An Australian-sovereign provider, ViewDS delivers innovative Identity, Credential, and Access Management (ICAM) solutions trusted globally by defence agencies, critical infrastructure operators, and enterprises with complex security needs. ViewDS solutions are deployed in over 30 countries, supporting high-security and performance-critical environments.

- **Access Sentinel**: Centralised, attribute-based access control for on-premises and cloud, using XACML for no-code policy management, adapting to evolving security needs.
- **Identity Bridge**: Synchronises data across on-premises and cloud, connecting diverse sources with custom transformations, notifications, and auditing for streamlined operations.
- **ViewDS Directory**: A secure, scalable directory service managing identity data with robust replication, high availability, and advanced search, ideal for high-security environments.
- **Cobalt**: Cloud-native platform for managing and securing access across hybrid environments, offering flexible identity management, policy enforcement, and real-time access synchronisation with integrated compliance tools.

To learn more about how ViewDS can support your Zero Trust and ICAM initiatives, visit www.viewds.com or contact us at sales@viewds.com.

About the Author

Ross Brenner is a Solution Architect at ViewDS Identity Solutions, specialising in identity, cybersecurity, and DevSecOps. With over 18-years experience across military, government, and enterprise environments, he brings a multidisciplinary approach to access control and Zero Trust initiatives. Ross is an AWS-Certified Solutions Architect and serves on the Melbourne branch committee of the Australian Information Security Association (AISA).

References

Scarfone, K.A. (2014). *NIST SP 800-162: Guide to Attribute Based Access Control (ABAC) Definition and Considerations*. National Institute of Standards and Technology (NIST).

Kindervag, J. et al. (2020). *NIST SP 800-207: Zero Trust Architecture*. National Institute of Standards and Technology (NIST).

Ferraiolo, D. et al. (2016). *NIST SP 800-178: Comparison of Attribute-Based Access Control (ABAC) Standards: XACML and NGAC*. National Institute of Standards and Technology (NIST).

Department of Defense Chief Information Officer (2022). *DoD Zero Trust Reference Architecture, Version 1.0*. U.S. Department of Defense.

NATO Communications and Information Agency. (2024). *NATO Digital Backbone Reference Architecture* (Draft). [Excerpt on Data-Centric Security and ABAC].

Styra, Inc. (2023). *OPA vs. XACML: Which is Better for Authorisation?* [Blog post].

Borja Roux Lorenzo. (2018). *ABAC and XACML: Where Does the Complexity Come From?* [LinkedIn Article].

ViewDS Identity Solutions. (2024). *Access Sentinel Policy Administration Guide*.

Abbreviations and Acronyms

ABAC	Attribute-Based Access Control
ADF	Australian Defence Force
APIs	Application Programming Interfaces
ASD	Australian Signals Directorate
CISO	Chief Information Security Officer
GUI	Graphical User Interface
ICAM	Identity, Credential, and Access Management
IdP	Identity Provider
LDAP	Lightweight Directory Access Protocol
NATO	North Atlantic Treaty Organisation
NGAC	Next Generation Access Control
NIST	National Institute of Standards and Technology
OPA	Open Policy Agent
PAP	Policy Administration Point
PDP	Policy Decision Point
PEP	Policy Enforcement Point
PIP	Policy Information Point
RBAC	Role-Based Access Control
SAML	Security Assertion Markup Language
STANAG	Standardisation Agreement
URN	Uniform Resource Name
XACML	eXtensible Access Control Markup Language
ZTA	Zero Trust Architecture