Sicherheitsarchitektur

Systemübersicht

Das Nago Framework implementiert eine mehrschichtige Sicherheitsarchitektur mit klarer Trennung von Verantwortlichkeiten.

C4 System Context Diagramm

```
@startuml
!include https://raw.githubusercontent.com/plantuml-stdlib/C4-
PlantUML/master/C4_Context.puml

title System Context - Nago Low-Code Framework

Person(user, "Benutzer", "Anwendungsbenutzer mit Browser")
Person(admin, "Administrator", "System-Administrator")

System(nago, "Nago Application", "Low-Code Framework\nGo Backend + VueJS Frontend")

System_Ext(nls, "Nago Login Service", "OAuth2/OIDC Provider\nSingle Sign-On")
System_Ext(smtp, "SMTP Server", "E-Mail-Versand\nVerifikation, Benachrichtigungen")
System_Ext(ai, "AI Provider", "Mistral AI, OpenAI\nOptionale KI-Integration")

Rel(user, nago, "Verwendet", "HTTPS/WebSocket")
Rel(admin, nago, "Administriert", "HTTPS/WebSocket")
Rel(nago, nls, "SSO Authentication", "HTTPS/OAuth2")
Rel(nago, smtp, "Sendet E-Mails", "TLS/SMTP")
Rel(nago, ai, "KI-Anfragen", "HTTPS/REST")

@enduml
```

C4 Container Diagramm

```
@startuml
!include https://raw.githubusercontent.com/plantuml-stdlib/C4-
PlantUML/master/C4_Container.puml

title Container Diagramm - Nago Application

Person(user, "Benutzer", "Browser")

System_Boundary(nago, "Nago Application") {
    Container(spa, "VueJS SPA", "TypeScript/Vue 3", "Single Page Application\nUI-
Rendering, State Management")
```

```
    Container(backend, "Go Backend", "Go 1.25", "Business Logic\nSession
Management\nPermission System")
    Container(ws, "WebSocket Handler", "gorilla/websocket", "Bidirektionale
Kommunikation\n/wire Endpoint")
    ContainerDb(blob, "Blob Storage", "File System", "Verschlüsselte
Datenspeicherung\nAES-GCM-256")
    Container(eventbus, "Event Bus", "Go Channels", "Publish-Subscribe\nAudit Events")
}

Rel(user, spa, "Lädt", "HTTPS")
Rel(spa, ws, "WebSocket", "wss://")
Rel(ws, backend, "Events", "Internal")
Rel(backend, blob, "Read/Write", "Encrypted")
Rel(backend, eventbus, "Publish", "Async")

@enduml
```

Komponenten-Diagramm

```
@startuml
!include https://raw.githubusercontent.com/plantuml-stdlib/C4-
PlantUML/master/C4_Component.puml

title Sicherheitskomponenten - Nago Backend

Container_Boundary(backend, "Go Backend") {
    Component(session, "Session Management", "application/session", "Cookie-basierte
Sessions\n3 Monate TTL\nHttpOnly/Secure")
    Component(user, "User Management", "application/user",
"Benutzerregistrierung\nArgon2id Hashing\nE-Mail-Verifikation")
    Component(permission, "Permission System", "application/permission", "Deklarative
Permissions\nAudit-Trail\nRBAC")
    Component(role, "Role Management", "application/role", "Rollenbasierte
Zugriffskontrolle\nPermission-Aggregation")
    Component(crypto, "Crypto Layer", "pkg/blob/crypto", "AES-GCM-256\nMaster Key
Management")
    Component(backup, "Backup System", "application/backup", "Verschlüsselte
Backups\nMaster Key Export/Import")
    Component(evs, "Event Store", "application/evs", "Audit Trail\nImmutable Events")
}

Rel(session, user, "Authentifiziert")
Rel(user, permission, "Prüft Berechtigungen")
Rel(permission, role, "Aggregiert aus Rollen")
Rel(user, crypto, "Passwort-Hashing")
Rel(backup, crypto, "Verschlüsselung")
Rel(permission, evs, "Audit Events")
```

```
@enduml
```

Authentifizierung

Session Management

Das Session Management basiert auf Cookie-basierter Authentifizierung mit folgenden Eigenschaften:

Eigenschaft	Wert	Sicherheitsrelevanz
Cookie-Name	`wdy-ora-access`	Nicht erratbar, proprietär
TTL	3 Monate nach letzter Authentifizierung	Langlebige Sessions für UX
HttpOnly	Ja	XSS-Schutz
Secure	Ja (Production)	MITM-Schutz
SameSite	Strict	CSRF-Schutz

Code-Referenz: Session-Erstellung

```go
// application/session/usecases.go:31
type Login func(id ID, login user.Email, password user.Password) (bool, error)
```

Passwort-Authentifizierung

Passwörter werden mit **Argon2id** gehasht, dem von OWASP empfohlenen Algorithmus:

Parameter	Wert	OWASP-Empfehlung
Algorithmus	Argon2id	Empfohlen
Memory	19 MiB	Minimum 19 MiB
Iterationen	2	Minimum 2
Parallelität	1	Minimum 1
Schlüssellänge	32 Byte	Standard

Code-Referenz: Argon2id-Parameter

```go
// application/user/password.go:111-113
// OWASP Settings:
https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
// Use Argon2id with minimum configuration of 19 MiB of memory, iteration count of 2,
and 1 degree of parallelism.
func argon2idMin(password string, salt []byte) []byte {
```

```
    return argon2.IDKey([]byte(password), salt, 2, 19*1024, 1, 32)
}
```

Passwort-Stärke-Validierung

Die Passwort-Validierung folgt BSI- und OWASP-Empfehlungen:

Kriterium	Anforderung	Status
Minimale Länge	8 Zeichen	✅ Implementiert
Maximale Länge	1000 Zeichen	✅ DoS-Schutz
Sonderzeichen	Mindestens 1	✅ Implementiert
Zahlen	Mindestens 1	✅ Implementiert
Groß-/Kleinschreibung	Beide erforderlich	✅ Implementiert
Entropie-Score	> 60 Bits	✅ Implementiert

Code-Referenz: Passwort-Validierung

```
// application/user/password.go:156-205
func CalculatePasswordStrength(p string) PasswordStrengthIndicator {
    var res PasswordStrengthIndicator
    res.MinLengthRequired = 8
    res.MaxLengthRequired = 1000  // DoS-Schutz
    // ...
    res.Acceptable = res.ContainsMinLength && res.ContainsSpecial &&
                     res.ContainsBelowMaxLength && res.ContainsNumber &&
                     res.ContainsUpperAndLowercase && res.Complexity > Weak
}
```

Security Note aus `application/user/password.go:49`:

> "security note: we must not introduce another sleep here, because argon2id should be safe enough and we must not stall any locks here. We need a totally different attack mitigation strategy."

Argon2id bietet durch seinen Memory-Hard-Charakter ausreichenden Schutz gegen Timing-Attacken.

E-Mail-Verifikation

Die E-Mail-Verifikation ist **verpflichtend** für die Anmeldung:

Code-Referenz: E-Mail-Verifikationspflicht

```
// application/user/uc_authenticates_by_password.go:42-47
if !usr.EMailVerified {
    // security note: intentionally it is not safe to let the user login, if his EMail
was never
    // verified. This opens up all kinds of identity stealing by default, even though
this may
    // be common in the world of shopping systems
    return std.None[User](), std.NewLocalizedError("Login nicht möglich",
        "Das Konto muss zuerst bestätigt werden").WithError(EMailNotVerifiedErr)
}
```

Single Sign-On (SSO)

SSO wird über den **Nago Login Service (NLS)** bereitgestellt:

```
@startuml
title SSO Authentication Flow

actor User
participant "Nago App" as App
participant "NLS Provider" as NLS

User -> App: Login-Anfrage
App -> App: StartNLSFlow()
App -> NLS: OAuth2 Authorization Request
NLS -> User: Login-Formular
User -> NLS: Credentials
NLS -> App: Authorization Code
App -> App: ExchangeNLS()
App -> NLS: Token Exchange
NLS -> App: Access + Refresh Token
App -> App: MergeSingleSignOnUser()
App -> User: Authenticated Session

@enduml
```

Bootstrap-Admin-Mechanismus

Für initiale Systemkonfiguration existiert ein temporärer Admin-Zugang:

Code-Referenz: Bootstrap Admin mit Zeitlimit

```
// application/management_user.go (via EnableBootstrapAdmin)
// Der Bootstrap-Admin ist zeitlich begrenzt und wird nach Ablauf automatisch
deaktiviert.
```

```
// Alle Aktionen werden im Audit-Log erfasst.
```

Der Bootstrap-Admin sollte nur für die initiale Konfiguration verwendet werden. Nach der Einrichtung regulärer Administratoren sollte dieser deaktiviert werden.

Autorisierung

Role-Based Access Control (RBAC)

Das RBAC-System basiert auf drei Ebenen:

```
@startuml
title RBAC-Modell

class User {
    +ID: string
    +Roles: []RoleID
    +Groups: []GroupID
}

class Role {
    +ID: string
    +Name: string
    +Permissions: []PermissionID
}

class Permission {
    +ID: string
    +Description: string
}

class Group {
    +ID: string
    +Name: string
    +Roles: []RoleID
}

User "1" -- "*" Role : hat
User "1" -- "*" Group : Mitglied von
Group "1" -- "*" Role : gewährt
Role "1" -- "*" Permission : enthält

@enduml
```

Permission-Audit

Jeder Berechtigungszugriff wird über die `Audit`-Funktion geprüft:

Code-Referenz: Permission-Audit

```
// application/permission/auditable.go
type Auditable interface {
    Audit(permission ID) error
    HasPermission(permission ID) bool
    AuditResource(name string, id string, p ID) error
    HasResourcePermission(name string, id string, p ID) bool
}
```

Privilege Escalation Schutz

Security Note aus `application/flow/evt_repository_assigned.go:27`:

> "security note: this is very important to keep: without this check, an attacker with only flow privileges can escalate privileges by assigning repositories with names of the system (like user, roles, groups) and mutate them in arbitrary ways."

Repository-IDs dürfen nicht mit `nago.` beginnen, um Privilege Escalation zu verhindern.

Code-Referenz: Repository-ID-Validierung

```
// application/flow/evt_repository_assigned.go:20-31
func (id RepositoryID) Validate() error {
    s := string(id)
    if strings.HasPrefix(s, "nago.") {
        // Schutz gegen Privilege Escalation
        return fmt.Errorf("repository id cannot start with 'nago.'")
    }
    // ...
}
```

Kryptographie

Verschlüsselungsarchitektur

```
@startuml
title Kryptographische Schichten
```

```
package "Application Layer" {
    [Password Hashing] as PH
    [Session Tokens] as ST
    [API Tokens] as AT
}

package "Storage Layer" {
    [Encrypted Blob Store] as EBS
    [Plain Blob Store] as PBS
}

package "Key Management" {
    [Master Key] as MK
    [Per-Entry Nonces] as PN
}

PH --> [Argon2id]
ST --> [Random Generation]
AT --> [Argon2id without Salt]

EBS --> MK
EBS --> PN
MK --> [AES-GCM-256]
PN --> [AES-GCM-256]

@enduml
```

AES-GCM-256 Implementierung

Code-Referenz: Verschlüsselungsalgorithmus

```
// pkg/blob/crypto/crypto.go:34-56
// Provides symmetric authenticated encryption using 256-bit AES-GCM with a random
nonce.

func encrypt(plaintext []byte, key *[32]byte) (ciphertext []byte, err error) {
    block, err := aes.NewCipher(key[:])
    if err != nil {
        return nil, err
    }

    gcm, err := cipher.NewGCM(block)
    if err != nil {
        return nil, err
    }

    nonce := make([]byte, gcm.NonceSize())
    _, err = io.ReadFull(rand.Reader, nonce)
    if err != nil {
```

```
        return nil, err
    }

    return gcm.Seal(nonce, nonce, plaintext, nil), nil
}
```

Master Key Management

Der Master Key ist ein 32-Byte (256-Bit) kryptographischer Schlüssel:

Quelle	Konfiguration	Sicherheit
Umgebungsvariable	`NAGO_MASTER_KEY` (Hex-encoded)	□ Empfohlen für Production
Lokale Datei	`.masterkey` mit 0600 Permissions	□□ Fallback, nicht für Production

Code-Referenz: Master Key Resolution

```
// application/security.go:27-67
func (c *Configurator) MasterKey() (crypto.EncryptionKey, error) {
    hexKey := os.Getenv(envNagoMasterKey)
    if hexKey := strings.TrimSpace(hexKey); hexKey != "" {
        // Aus Umgebungsvariable laden
        buf, err := hex.DecodeString(hexKey)
        // ...
    }

    // Fallback: Datei mit 0600 Permissions
    if err := os.WriteFile(c.masterKeyFile(),
        []byte(hex.EncodeToString((*key)[:])), 0600); err != nil {
        // ...
    }
}
```

Der Master Key muss **separat** vom Backup gespeichert werden. Bei Verlust des Master Keys können verschlüsselte Daten nicht wiederhergestellt werden.

Input Validation

User Enumeration Protection

Security Note aus `application/management_mail.go:126`:

> "security note: intentionally do not expose this information to the frontend"

> Bei Password-Reset-Anfragen wird nicht offengelegt, ob der Benutzer existiert.

Code-Referenz: User Enumeration Schutz

```
// application/management_mail.go:125-129
if optUser.IsNone() {
    // security note: intentionally do not expose this information to the frontend
    slog.Error("shall send verification mail but user not found", "mail", mail)
    return nil  // Kein Fehler zurückgeben
}
```

Bewusste User-Existenz-Offenlegung bei Registrierung

Security Note aus `application/user/uc_create.go:130`:

> "security note: this allows to expose the fact, that a user already exists in the system. however, there is no reasonable way to avoid that. We must not delay this with a sleep, because we are still in the write mutex lock and therefore would cause a kind of accumulating 'deadlock'/DOS like behavior"

Bei der Registrierung wird bewusst offengelegt, wenn eine E-Mail bereits existiert. Dies ist ein akzeptierter Trade-off zwischen UX und Security.

DoS-Schutzmaßnahmen

Mutex-Lock für User-Erstellung

Code-Referenz: Rate-Limiting bei User-Erstellung

```
// application/user/uc_create.go:31-32
// this is really harsh and allows intentionally only to create one user per second
mutex.Lock()
defer mutex.Unlock()
```

Passwort-Längenbegrenzung

Code-Referenz: DoS-Schutz durch maximale Passwortlänge

```
// application/user/password.go:167-170
// see
https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html
if strlen <= res.MaxLengthRequired {  // 1000 Zeichen
    // probably a DOS attack
    res.ContainsBelowMaxLength = true
```

```
}
```

Staging-ID-Sicherheit

Security Note aus `application/dataimport/ui/page_select_parser.go:72`:

> "security note: do not extend this by accepting existing staging id by query because our use cases have no security concept, which otherwise protects against inserting malicious data in foreign staging data sets."

Staging-IDs werden nicht via Query-Parameter akzeptiert, um Dateninjection zu verhindern.

Netzwerksicherheit

CORS-Konfiguration

Code-Referenz: CORS im Debug-Modus

```
// application/cfg_page.go:206-217
if c.debug {
    r.Use(
        cors.Handler(cors.Options{
            AllowedOrigins:   []string{"http://*"},  // NUR DEBUG!
            AllowedMethods:   []string{"GET", "POST", "PUT", "DELETE", "OPTIONS"},
            AllowedHeaders:   []string{"Accept", "Authorization", "Content-Type", "X-
CSRF-Token"},
            ExposedHeaders:   []string{"Link"},
            AllowCredentials: true,
            MaxAge:           300,
        }),
    )
    c.defaultLogger().Warn("using debug cors settings")
}
```

Die Debug-CORS-Einstellungen (`AllowedOrigins: ["http://"]`) dürfen *niemals* in Production verwendet werden!

WebSocket Security

Code-Referenz: WebSocket Origin-Validierung

```
// application/cfg_page.go:507-510
```

```
var upgrader = websocket.Upgrader{
    CheckOrigin: func(r *http.Request) bool {
        return true //TODO security implications?
    },
    EnableCompression: true,
}
```

TODO: Die WebSocket Origin-Validierung ist aktuell deaktiviert (`CheckOrigin: true`). Dies sollte für Production mit einer strikten Origin-Prüfung implementiert werden.

TLS für SMTP

Code-Referenz: TLS-Konfiguration für SMTP

```
// application/mail/smtp.go:28-31
tlsconfig := &tls.Config{
    InsecureSkipVerify: false,  // Zertifikatsvalidierung aktiviert
    ServerName:         host,
}
```

Logging und Audit

Strukturiertes Logging

Das Framework verwendet Go's `slog` für strukturiertes Logging:

```
slog.Info("user created",
    slog.String("userID", user.ID),
    slog.String("email", user.Email),
    slog.Time("createdAt", user.CreatedAt))
```

Event-basierter Audit Trail

Alle sicherheitsrelevanten Ereignisse werden über den Event Bus publiziert:

Code-Referenz: User-Created Event

```
// application/user/uc_create.go:157-166
eventBus.Publish(Created{
    ID:                user.ID,
    Firstname:         user.Contact.Firstname,
    Lastname:          user.Contact.Lastname,
    Email:             user.Email,
    PreferredLanguage: tag,
```

```
    NotifyUser:        notify,
    VerificationCode:  user.VerificationCode,
    CreatedAt:         user.CreatedAt,
})
```

Audit-UI

Die Audit-Funktionalität ist über das Admin Center zugänglich:

Code-Referenz: Audit Page

```
// application/evs/ui/page_audit.go:35-44
func PageAudit[Evt any](wnd core.Window, uc evs.UseCases[Evt], opts PageAuditOptions
[Evt]) core.View {
    // Zeigt alle Events mit Zeitstempel, User, Aktion
}
```