Atomik Security & Trust Document

For Security Professionals & Penetration Testers

Version: 1.0
Last Updated: January 2025
Contact: security@atomik.sh

Executive Summary

Atomik is a penetration testing report generation platform designed with security-first principles. This document provides transparency into our security architecture, data handling practices, and compliance posture to address concerns from security professionals who handle sensitive client vulnerability data.

Key Points:

- Multi-tenant architecture with strict data isolation
- Comprehensive audit logging of all actions
- Defense-in-depth security controls
- On-premise deployment option (coming Q2 2026)
- Zero-knowledge architecture (we cannot access your data)

1. Data Security & Encryption

1.1 Encryption in Transit

All data transmission is encrypted using TLS 1.2+:

- **API Communication**: All API endpoints require HTTPS (TLS 1.2+)
- **Database Connections**: PostgreSQL connections use SSL/TLS encryption
  - Configurable: `DATABASE_SSL_MODE` (require, verify-ca, verify-full)
  - Default: `prefer` (upgrades to `require` in production)
- **WebSocket Connections**: Encrypted via WSS (if used)
- **File Uploads**: All file uploads transmitted over HTTPS

Implementation:

- FastAPI enforces HTTPS in production

- Nginx reverse proxy terminates TLS
- Certificate pinning for critical services (configurable)

1.2 Encryption at Rest

Database Encryption:

- PostgreSQL supports at-rest encryption (enabled via database configuration)
- Database files encrypted at filesystem level (when using encrypted volumes)
- Sensitive fields can be encrypted using Fernet (AES-128) before storage

File Storage Encryption:

- **S3 Option**: Server-side encryption (SSE-S3 or SSE-KMS) when using AWS S3
- **Local Storage**: Files stored in encrypted volumes (customer-controlled)
- **Evidence Files**: Stored with restricted permissions (600) - owner read/write only

Password Storage:

- Passwords hashed using **bcrypt** with cost factor 12
- Never stored in plaintext
- Salt automatically generated per password
- Implementation: `passlib` with `bcrypt` backend

Code Reference:

```python
# backend/app/core/security.py
pwd_context = CryptContext(schemes=["bcrypt"],
deprecated="auto")
# Cost factor 12 = ~250ms hash time (prevents brute force)
```

1.3 Key Management

- **JWT Secret Keys**: Stored in environment variables (never in code)
- **Database Credentials**: Stored in environment variables
- **API Keys**: Customer-provided (OpenAI, etc.) - never logged or exposed
- **Session Tokens**: Cryptographically secure (UUID v4) with expiration

2. Multi-Tenancy & Data Isolation

2.1 Organization-Based Isolation

Every resource is scoped to an organization:

- **Clients**: `organizationId` foreign key - users can only access clients in their org
- **Projects**: Linked to clients, inheriting organization isolation
- **Findings**: Linked to projects, inheriting organization isolation
- **Reports**: Linked to projects, inheriting organization isolation
- **Users**: Bound to single organization via `organizationId`

Database Schema (Example):

```
None
model Organization {
  id   String @id
  // ... other fields
  users       User[]
  clients     Client[]
  projects    Project[]  // via Client
  templates   Template[]
}

model Client {
  organizationId String
  organization   Organization @relation(...)
  // Users can only access clients where client.organizationId
== user.organizationId
}
```

2.2 Access Control Enforcement

All API endpoints enforce organization-level access control:

Example from Projects API:

```
Python
# backend/app/api/routes/projects.py
if current_user.organizationId and client.organizationId !=
current_user.organizationId:
    raise HTTPException(
```

```
        status_code=status.HTTP_403_FORBIDDEN,
        detail="Access denied"
    )
```

Enforcement Points:

- **Project Access**: Verified before any project operation
- **Client Access**: Verified before client operations
- **Finding Access**: Verified via project → client → organization chain
- **Report Access**: Verified via project → client → organization chain

Result: Users **cannot** access data from other organizations, even if they know resource IDs.2.3 Database-Level Isolation

- **Row-Level Security**: Enforced at application level (Prisma queries)
- **No Cross-Organization Queries**: All queries include `organizationId` filter
- **Database User Permissions**: Application uses single database user (connection pooling)
- **Isolation Guarantee**: Application logic prevents cross-org access

3. Authentication & Authorization

3.1 Authentication Methods

Current (SaaS):

- Clerk JWT tokens with cryptographic verification (JWKS)
- Token signature verified on every request
- No token = no access

On-Premise (Coming Q2 2026):

- Password-based authentication (bcrypt hashed)
- JWT tokens with configurable expiration
- Optional: LDAP/Active Directory integration

3.2 Password Security

Password Policy (Enforced):

- **Minimum Length**: 12 characters
- **Complexity**: Uppercase, lowercase, numbers, special characters
- **Common Passwords**: Blocked (top 1000 common passwords)
- **Sequential Patterns**: Blocked (e.g., "1234", "abcd")
- **Repeated Characters**: Blocked (e.g., "aaaa")

Implementation:

```python
# backend/app/core/password_policy.py
MIN_PASSWORD_LENGTH = 12
REQUIRE_UPPERCASE = True
REQUIRE_LOWERCASE = True
REQUIRE_NUMBERS = True
REQUIRE_SPECIAL = True
```

Account Lockout:

- **Failed Attempts**: 5 failed logins
- **Lockout Duration**: 15 minutes
- **IP-Based Tracking**: Prevents distributed brute force
- **Audit Logged**: All failed attempts logged

3.3 Session Management

Session Security:

- **Maximum Sessions**: 5 concurrent sessions per user
- **Session Timeout**: 8 hours absolute
- **Idle Timeout**: 30 minutes
- **Token Expiration**: 30 minutes (access token), 7 days (refresh token)
- **Session Revocation**: On password change, logout, or admin action

Session Tracking:

- Session IDs stored securely
- IP address and user agent tracked
- Suspicious activity (IP change) can trigger re-authentication

3.4 Role-Based Access Control (RBAC)

Roles:

- **ADMIN**: Full access within organization
- **USER**: Standard access (create/edit findings, projects)
- **VIEWER**: Read-only access

Enforcement:

- Roles checked at API level
- Frontend UI respects roles (UX only - backend enforces)
- Role changes logged in audit trail

4. Input Validation & Injection Prevention

4.1 SQL Injection Prevention

Method: Prisma ORM with parameterized queries

- **All database queries** use Prisma (no raw SQL)
- **Parameterized queries** prevent SQL injection
- **Type-safe queries** prevent injection via type coercion

Example:

```python
# Safe - Prisma handles parameterization
user = await db.user.find_unique(where={"email": user_email})
# user_email is automatically escaped/parameterized
```

No Raw SQL: Codebase contains zero raw SQL queries.4.2 XSS (Cross-Site Scripting) Prevention

Frontend:

- React automatically escapes user input
- Rich text editor (TipTap) sanitizes HTML
- DOMPurify used for additional sanitization

Backend:

- **Input Sanitization**: All user input validated and sanitized
- **HTML Sanitization**: Bleach library for HTML content
- **Output Encoding**: All API responses properly encoded

Content Security Policy (CSP):

```python
# backend/app/core/security_middleware.py
"Content-Security-Policy": "default-src 'self'; script-src
'self' 'unsafe-inline'; ..."
```

4.3 Command Injection Prevention

File Operations:

- All file paths validated for path traversal (`../` blocked)
- Filenames sanitized (alphanumeric, dash, underscore, dot only)
- Command arguments validated before subprocess calls

Implementation:

```python
# backend/app/core/input_validation.py
PATH_TRAVERSAL_PATTERN = re.compile(r'\.\./|\.\.\\')
COMMAND_INJECTION_PATTERN = re.compile(r'[;&|`$()]')
```

4.4 File Upload Security

Multi-Layer Validation:

1. **Extension Whitelist**: Only allowed extensions (png, jpg, pdf, xml, etc.)
2. **Magic Byte Verification**: File content verified (not just extension)
3. **File Size Limits**: 10MB maximum per file
4. **SVG Sanitization**: Removes embedded scripts, event handlers, javascript: URLs
5. **MIME Type Validation**: Content-type verified against file content

Implementation:

```python
# backend/app/core/file_validation.py
# Validates file type by examining file content (magic bytes)
# Prevents: evil.php uploaded as evil.png
```

SVG Security:

- `<script>` tags removed
- Event handlers (`onclick`, `onerror`, etc.) removed
- `javascript:` URLs removed
- `<foreignObject>` removed (can embed HTML/JS)

5. Audit Logging & Compliance

5.1 Comprehensive Audit Trail

All actions are logged:

- **User Actions**: CREATE, READ, UPDATE, DELETE on all resources
- **Authentication Events**: Login success/failure, logout, token refresh
- **Security Events**: Rate limiting, access denied, invalid input
- **Data Exports**: Report generation, data exports logged
- **Admin Actions**: Role changes, settings changes

Audit Log Fields:

- User ID and email
- Action type

- Resource type and ID
- IP address
- User agent
- Timestamp
- Success/failure status
- Error messages (if failed)

Database Model (Example):

```
None
model AuditLog {
  id          String
  timestamp   DateTime
  userId      String?
  userEmail   String?
  action      AuditAction  // CREATE, READ, UPDATE, DELETE,
etc.
  resource    String       // "Finding", "Project", "Client"
  resourceId  String?
  ipAddress   String?
  userAgent   String?
  success     Boolean
  errorMsg    String?
  // ... indexed for fast queries
}
```

5.2 Compliance Readiness

GDPR Compliance:

- User data export: `/api/users/export` (all user data)
- User data deletion: `/api/users/delete` (right to be forgotten)
- Data minimization: Only collect necessary data
- Consent tracking: User consent logged

SOC 2 Readiness:

- Access controls documented and enforced
- Audit logging comprehensive
- Security monitoring in place
- Incident response plan documented

OWASP Top 10:2025 Coverage:

- A01: Broken Access Control → Organization-level isolation enforced
- A02: Cryptographic Failures → TLS 1.2+, bcrypt passwords

- A03: Injection → Prisma ORM, input validation
- A04: Insecure Design → Security-first architecture
- A05: Security Misconfiguration → Security headers, secure defaults
- A06: Vulnerable Components → Dependency scanning
- A07: Authentication Failures → Strong passwords, session management
- A08: Software & Data Integrity → Signature verification, audit logs
- A09: Logging Failures → Comprehensive audit logging
- A10: SSRF → URL validation, IP filtering

6. Architecture Security

6.1 Defense in Depth

Multiple Security Layers:

1. **Network Layer**: TLS encryption, firewall rules
2. **Application Layer**: Authentication, authorization, input validation
3. **Database Layer**: Parameterized queries, connection encryption
4. **File System Layer**: File permissions, path validation
5. **Monitoring Layer**: Audit logs, rate limiting, anomaly detection

6.2 Security Headers

All responses include security headers:

```
Python
# backend/app/core/security_middleware.py
- Strict-Transport-Security: max-age=31536000;
includeSubDomains
- Content-Security-Policy: Prevents XSS and injection
- X-Content-Type-Options: nosniff
- X-Frame-Options: SAMEORIGIN
- X-XSS-Protection: 1; mode=block
- Referrer-Policy: strict-origin-when-cross-origin
- Permissions-Policy: Restricts browser features
```

6.3 Rate Limiting

Protection against abuse:

- **Auth Endpoints**: 20 requests/minute
- **File Uploads**: 30 requests/minute
- **AI Endpoints**: 20 requests/minute
- **Scan Imports**: 10 requests/minute
- **Default**: 60 requests/minute

Implementation:

- Redis-based (production) or in-memory (development)
- Sliding window algorithm
- Per-IP and per-user tracking
- Rate limit violations logged

6.4 Error Handling

Security-Focused Error Messages:

- **Production**: Generic error messages (no stack traces)
- **No Information Leakage**: Errors don't reveal system internals
- **Request IDs**: Included for support (not exposed to attackers)
- **Detailed Logging**: Full error details logged server-side only

7. Data Sovereignty & On-Premise Option

7.1 Current Architecture

SaaS Deployment:

- Hosted on Railway (backend) and Vercel (frontend)
- Data stored in PostgreSQL (Railway managed)
- Files stored in Docker volumes or S3 (customer choice)
- **We cannot access your data** (no backdoors, no admin access to customer databases)

7.2 On-Premise Deployment **(Coming Q2 2026)**

Full Control for Enterprise:

- **Deployment**: Docker containers (no source code)
- **Data Location**: Your servers, your control
- **Network**: Air-gapped deployment supported
- **License Validation**: Online or offline (encrypted license files)
- **Updates**: Customer-controlled (pull from private registry)

Benefits:

- Complete data sovereignty
- No external dependencies
- Compliance with strict regulations (government, defense)
- Custom integrations possible
- Full audit trail on your infrastructure

Timeline: Q2 2026 (in development)

7.3 Data Residency

Current (SaaS):

- Database: Railway (US-based, configurable region)
- Files: Customer choice (local volumes or S3 in any region)

On-Premise:

- **100% customer-controlled**
- Data never leaves your infrastructure
- No external API calls (except optional license validation)

8. Third-Party Dependencies

8.1 Authentication (Clerk)

Current SaaS:

- Clerk handles authentication (industry-standard)
- JWKS verification (cryptographic signature verification)
- We never see or store passwords
- Token verification happens on every request

On-Premise:

- Clerk removed
- Password-based auth (bcrypt)
- Optional: LDAP/AD integration

8.2 AI Features (OpenAI)

Optional Feature:

- Customer provides their own OpenAI API key
- API key stored in environment variable (never logged)
- Requests go directly from your instance to OpenAI
- We never see API keys or AI requests
- **Can be completely disabled** (no API key = AI features disabled)

On-Premise:

- Customer controls OpenAI API key
- Or use self-hosted LLM (Ollama, LocalAI) - coming soon
- Or disable AI features entirely

8.3 Payment Processing (Paddle)

SaaS Only:

- Paddle handles payments (PCI-compliant)

- Webhook signatures verified cryptographically
- We never see credit card data

On-Premise:

- Paddle removed
- License-based (no payment processing needed)

9. Security Monitoring & Incident Response

9.1 Security Monitoring

Automated Monitoring:

- Failed login attempts tracked
- Rate limit violations logged
- Access denials logged with context
- Invalid input attempts logged
- Unusual activity patterns detected

Alerting:

- Multiple failed logins → Alert
- Excessive access denials → Alert
- Rate limit violations → Alert
- Security event patterns → Alert

9.2 Incident Response

Process:

1. **Detection**: Automated alerts or manual reports
2. **Triage**: Assess severity and impact
3. **Containment**: Isolate affected systems
4. **Investigation**: Review audit logs and traces
5. **Remediation**: Apply fixes and patches
6. **Communication**: Notify affected users (if required)
7. **Post-Incident**: Document and improve

Response Times:

- Critical: Immediate
- High: Within 4 hours
- Medium: Within 24 hours
- Low: Within 7 days

10. Security Testing & Validation

10.1 Security Testing

Regular Testing:

- Dependency vulnerability scanning (weekly)
- Code security reviews (pre-commit)
- Security audits (quarterly)
- Penetration testing (annual - planned)

Tools Used:

- `safety` (Python dependency scanning)
- `pip-audit` (Python vulnerability scanning)
- `npm audit` (JavaScript dependency scanning)
- Manual code reviews

10.2 Security Certifications (Planned)

Roadmap:

- SOC 2 Type II (2025)
- ISO 27001 (2025-2026)
- Penetration testing reports (available on request)

11. Data Handling & Privacy

11.1 What We Store

Customer Data:

- Penetration test findings and reports
- Client information
- Project data
- Evidence files (screenshots, attachments)
- User accounts and authentication data

What We Don't Store:

- Credit card information (handled by Paddle)
- Passwords in plaintext (bcrypt hashed only)
- API keys (customer-provided, stored in env vars)

11.2 Data Access

Our Access:

- **We cannot access your data** in SaaS mode (no admin backdoors)
- Database credentials are customer-managed (Railway)
- Application code has no hardcoded credentials
- Support access requires explicit customer permission

On-Premise:

- **Zero access** - completely customer-controlled
- No external connections (except optional license validation)

11.3 Data Retention

Customer-Controlled:

- Data retention policies set by customer
- Data deletion available via API
- Export functionality for data portability
- Backup retention: Customer-controlled

12. Trust & Transparency

12.1 Open Security Practices

We Practice:

- Security-first development
- Defense-in-depth architecture
- Comprehensive audit logging
- Regular security updates
- Transparent security documentation

12.2 Security Contact

For Security Concerns:

- Email: **security@atomik.sh**
- Response Time: Within 48 hours
- Responsible Disclosure: Appreciated and rewarded

12.3 Security Documentation

Available Documentation:

- This security document
- API security documentation
- Architecture diagrams (available on request)
- Compliance readiness documentation

13. On-Premise Security Advantages

13.1 Complete Control

With On-Premise Deployment:

- **Data Never Leaves Your Infrastructure**: 100% data sovereignty
- **No External Dependencies**: Air-gapped deployment possible
- **Your Security Policies**: Enforce your own security controls
- **Your Compliance**: Meet your specific compliance requirements
- **Your Monitoring**: Integrate with your SIEM/SOC

13.2 Security Benefits

- **Network Isolation**: Deploy in isolated network segments
- **Firewall Rules**: Your firewall, your rules
- **Intrusion Detection**: Your IDS/IPS systems
- **Backup Strategy**: Your backup and disaster recovery
- **Access Control**: Your LDAP/AD integration

14. Comparison: SaaS vs On-Premise

Feature	SaaS (Current)	On-Premise (Q2 2026)
Data Location	Railway (US)	Your servers
Data Access	We cannot access	You have full control
Network	Internet required	Air-gapped possible
Compliance	SOC 2 (planned)	Your compliance
Updates	Automatic	Customer-controlled
Customization	Limited	Full customization
External APIs	Clerk, Paddle	None (optional license)
AI Features	Optional (OpenAI)	Optional (OpenAI or self-hosted)

15. Recommendations for Security-Conscious Organizations

15.1 For Maximum Security

Recommended Approach:

1. **Use On-Premise Deployment** (available **Q2 2026**)
2. **Deploy in Isolated Network**: Air-gapped or VPN-only
3. **Disable AI Features**: If not needed, don't provide OpenAI key
4. **Enable Audit Logging**: Review logs regularly
5. **Use Strong Passwords**: Enforce organization password policy
6. **Regular Backups**: Implement your backup strategy
7. **Monitor Access**: Review audit logs for suspicious activity

15.2 For SaaS Users (Current)

Security Best Practices:

1. **Use Strong Passwords**: 12+ characters, complex
2. **Enable MFA**: When available (planned)
3. **Review Audit Logs**: Regularly check for suspicious activity
4. **Limit User Access**: Principle of least privilege
5. **Regular Exports**: Backup your data regularly
6. **Monitor Access**: Watch for unusual login patterns

16. Technical Security Details

16.1 Code Security

Secure Coding Practices:

- Type-safe code (TypeScript + Python type hints)
- Input validation on all endpoints
- Parameterized queries (Prisma ORM)
- Secure defaults (fail-secure)
- No hardcoded secrets
- Environment variables for all configuration

16.2 Dependency Security

Dependency Management:

- Lock files committed (poetry.lock, package-lock.json)
- Regular vulnerability scanning
- Security patches applied within 7 days
- No known high/critical vulnerabilities

Key Dependencies:

- FastAPI (Python web framework)
- Prisma (Database ORM - type-safe)
- React (Frontend framework)
- bcrypt (Password hashing)
- jose (JWT handling)

16.3 Infrastructure Security

Deployment:

- Docker containers (isolated)
- Nginx reverse proxy (security headers)
- PostgreSQL (encrypted connections)
- Redis (for rate limiting, isolated)

Network:

- TLS 1.2+ required
- CORS properly configured
- Security headers enforced
- Rate limiting active

17. Frequently Asked Questions

Q: Can Atomik access my client's vulnerability data?

A: No. In SaaS mode, we cannot access your database (credentials are managed by Railway). In on-premise mode, you have complete control - we have zero access.

Q: Is my data encrypted?

A: Yes. All data in transit is encrypted (TLS 1.2+). Data at rest can be encrypted (database encryption, encrypted volumes). Passwords are hashed with bcrypt.

Q: Can other organizations see my data?

A: No. Strict organization-level isolation enforced at database and application level. Users can only access data within their organization.

Q: What happens if Atomik is breached?

A:

- In SaaS: Your database is separate (Railway-managed), so breach impact is limited
- In On-Premise: Your infrastructure, your security - breach would be on your side
- Audit logs would show any unauthorized access
- We have incident response procedures

Q: Can I export my data?

A: Yes. Data export functionality available via API. Full database backups available (customer-controlled).

Q: Is on-premise really coming?

A: Yes. **Q2 2026.** Development in progress. Will support:

- Docker deployment (no source code)
- License key validation
- Air-gapped deployment
- Full customer control

Q: What about compliance (SOC 2, ISO 27001)?

A:

- Security controls are SOC 2-ready (documented, enforced)
- SOC 2 Type II certification planned for 2026
- ISO 27001 planned for 2025-2026
- Current architecture follows best practices
- **Atomik was created by pentesters for pentesters, the entire application has been pentested by seasoned members of the Team.**

18. Conclusion

Atomik is built with security as a foundational principle, not an afterthought. We understand that penetration testers handle highly sensitive client data and have architected the platform accordingly.

Key Security Guarantees:

1. **Data Isolation**: Multi-tenant architecture with strict organization-level isolation
2. **Encryption**: TLS in transit, encryption at rest (configurable)
3. **Access Control**: Role-based access control enforced at API level
4. **Audit Logging**: Comprehensive logging of all actions
5. **Input Validation**: Defense against injection attacks
6. **On-Premise Option**: Complete data sovereignty (Q2 2026)

For the Most Security-Conscious:

- Wait for on-premise deployment (Q2 2026)
- Deploy in your own infrastructure
- Maintain complete control
- Meet your specific compliance requirements

For Current SaaS Users:

- Your data is isolated and encrypted
- We cannot access your database
- Comprehensive audit logging
- Security-first architecture

Contact & Support

Security Concerns: security@atomik.sh
General Support: support@atomik.sh
On-Premise Inquiries: enterprise@atomik.sh

Response Times:

- Security issues: 48 hours
- General support: 24 hours
- Enterprise inquiries: 24 hours

-----*This document is updated regularly. Last updated: January 2025*