NEXUS PROTOCOL

Real-Time Cyber Warfare Game

TECHNICAL DESIGN DOCUMENT & COMPLETE WORKFLOW

Version 1.0 | February 2026 | CONFIDENTIAL

CONCEPT	A real-time cyber warfare simulation where Red Team players actively attack a live vulnerable system while Blue Team players actively defend the same system — simultaneously, in the same environment.

NOT CTF	NOT static challenges or hidden flags
NOT SCRIPTED	NOT predetermined outcomes or fake terminals
NOT SIMULATED	Real Linux systems, real tools, real consequences
IS WARFARE	Red attacks. Blue defends. Same battlefield. Same moment.

2 TEAMS Red + Blue	**1** BATTLEFIELD Shared Live VM	**REAL** TOOLS nmap metasploit sqlmap	**LIVE** SCORING Real actions only	**ADMIN** CONTROL Full monitor + control

TABLE OF CONTENTS

01 — GAME CONCEPT OVERVIEW

NEXUS PROTOCOL is a real-time cyber warfare simulation. It is NOT a Capture-The-Flag game. There are no hidden flag strings, no static puzzles, no scripted responses. Two teams — Red and Blue — fight over the same live virtual machine simultaneously. Red Team attacks using real offensive security tools. Blue Team defends using real monitoring and incident response techniques. Every action has real consequences on the shared environment.

RED TEAM — Offensive Operations

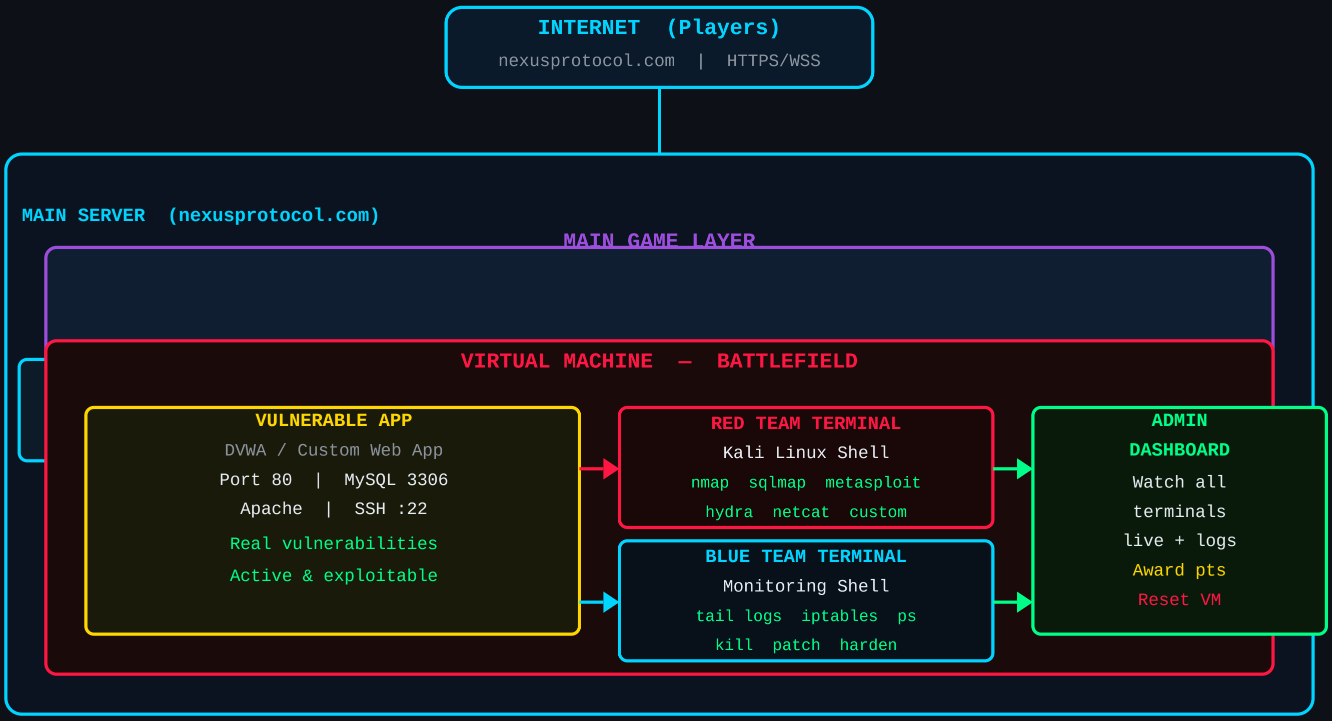
- Reconnaissance — nmap, masscan, dirb, OSINT gathering on the target
- Initial Access — SQL injection, RCE, password brute-force via hydra
- Exploitation — Metasploit modules, custom exploits, web shells
- Privilege Esc. — Kernel exploits, SUID misconfigs, sudo abuse
- Lateral Movement — Pivoting through network, pass-the-hash, credential reuse
- Persistence — Backdoors, cron jobs, SSH key injection
- Exfiltration — Database dumps, file exfil, C2 channels

BLUE TEAM — Defensive Operations

- Monitoring — tail logs, watch netstat, SIEM alerts, process monitoring
- Detection — Identify port scans, anomalous traffic, failed logins
- Response — iptables block, kill malicious process, isolate service
- Forensics — Investigate webshells, log correlation, timeline build
- Hardening — Patch vulnerabilities live, update firewall rules, ACLs
- Recovery — Restore from snapshot, remove backdoors, credential reset
- Documentation — Incident report, attack timeline, lessons learned

02 — SERVER ARCHITECTURE

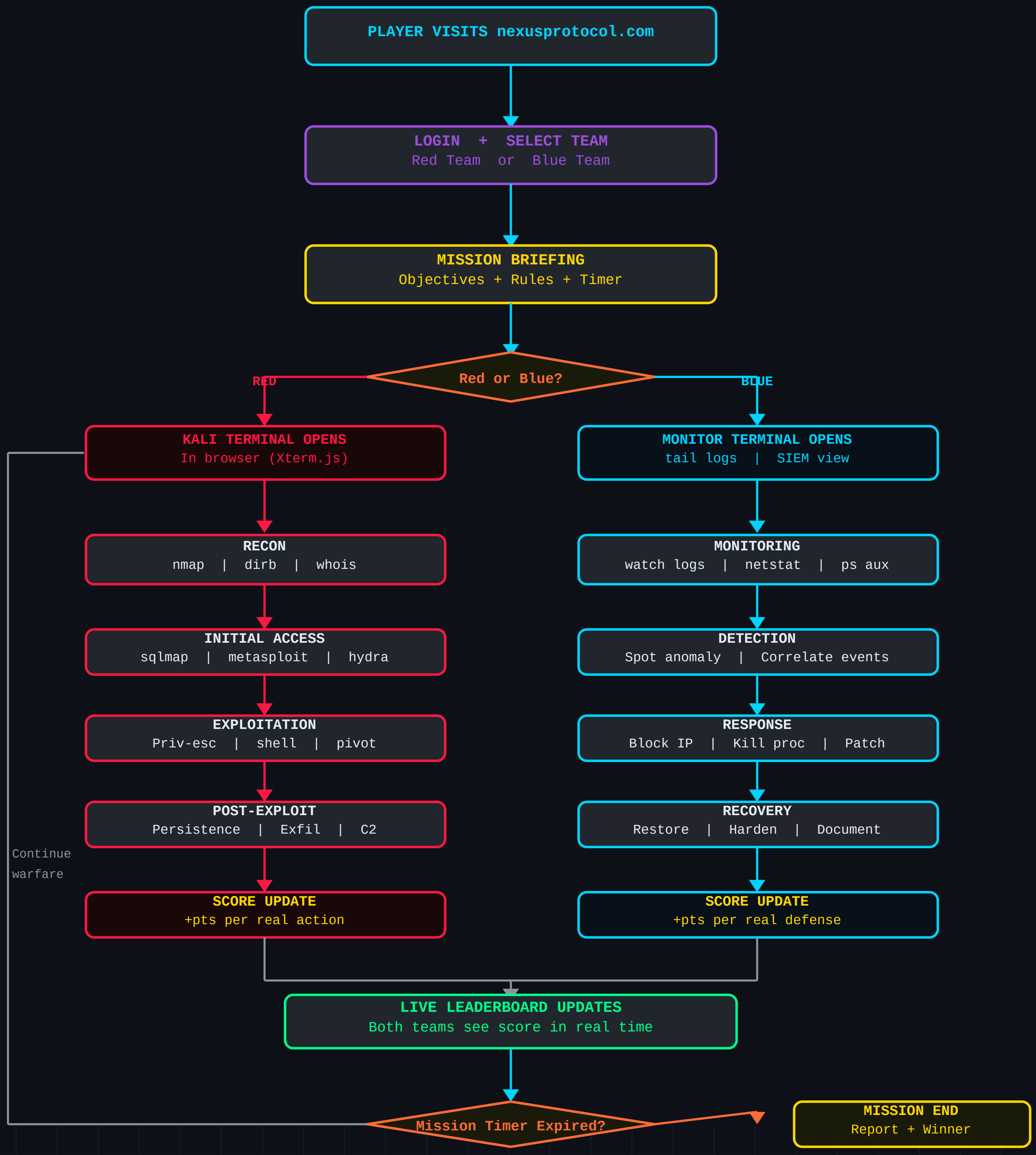
One physical main server hosts both the game platform and the virtual machine battlefield. The main game runs publicly at the domain. The VM runs internally and is accessed only through the SSH proxy built into the game backend — players never connect directly.



Component	Location	Purpose
Main Game	Main Server (public)	React UI + Node.js backend + WebSocket + DB
SSH Proxy	Main Server (internal)	Bridges browser terminal to VM shell
Vulnerable App	VM Port 80	DVWA or custom app — real vulnerabilities
Red Terminal	VM via SSH	Kali Linux shell with all attack tools
Blue Terminal	VM via SSH	Monitoring shell — logs, network, processes
Admin Dashboard	Main Server (auth)	Live mirrors of all terminals + controls

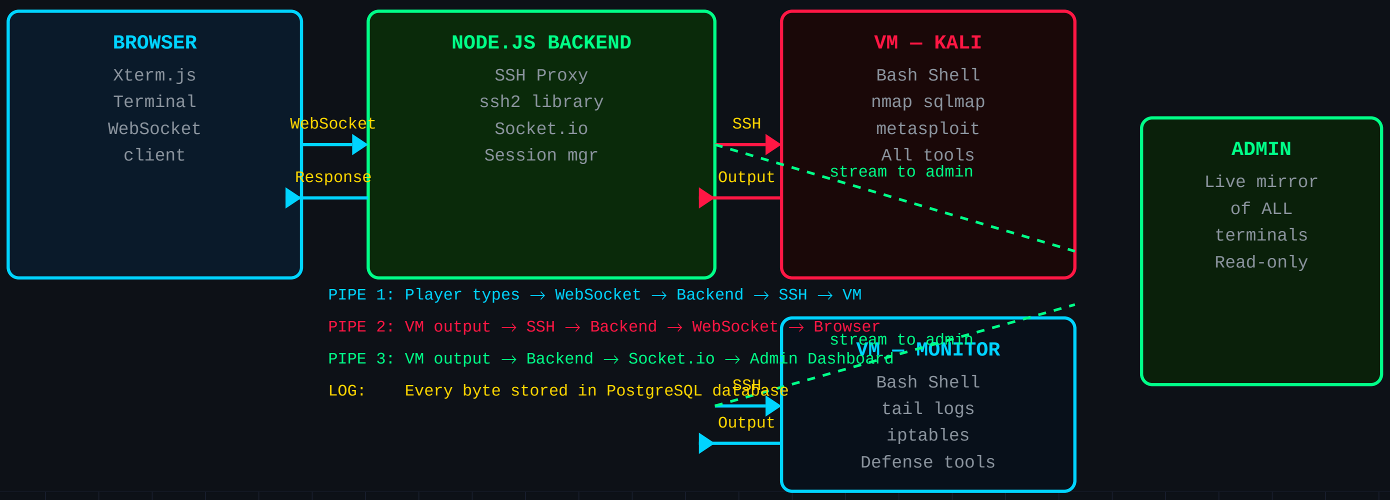
03 — PLAYER JOURNEY FLOWCHART

Both teams enter through the same game interface but diverge into their respective terminals after team selection. All actions happen concurrently on the shared VM battlefield.



04 — SSH TERMINAL INTEGRATION

The browser terminal is powered by Xterm.js rendering over a WebSocket. The Node.js backend acts as an SSH proxy using the ssh2 library, maintaining a persistent SSH session to the VM. Three simultaneous data pipes run for every active player session.



PIPE 1: Player types → WebSocket → Backend → SSH → VM

PIPE 2: VM output → SSH → Backend → WebSocket → Browser

PIPE 3: VM output → Backend → Socket.io → Admin Dashboard

LOG: Every byte stored in PostgreSQL database

Pipe	Direction	Purpose
PIPE 1	Browser → Backend → VM	Player keystrokes sent to VM shell
PIPE 2	VM → Backend → Browser	VM output shown in player terminal
PIPE 3	VM → Backend → Admin	All output mirrored to admin dashboard live
LOG	Backend → Database	Every byte stored for audit + replay

TECH: ssh2 npm library + Xterm.js + Socket.io

The ssh2 library opens an authenticated SSH connection from Node.js to the VM. A pseudoterminal (PTY) session is created for each player. Input/output streams are piped through Socket.io WebSocket to the Xterm.js renderer in the browser. The same stream is tee'd to the admin namespace and the database logger simultaneously.

05 — REAL-TIME SCORING ENGINE

Scoring is based entirely on real actions performed in the live environment. There are no flags to submit as the primary mechanic. The backend continuously monitors terminal output streams, VM logs, and system state to auto-detect achievements and fire scoring events in real time.

RED TEAM SCORING

Port scan done	+10
Initial access	+75
SQL injection ok	+80
Shell on target	+150
Root / admin	+200
Sensitive file read	+100
Data exfiltrated	+175
Persistence set	+200
Lateral movement	+150
Stayed undetected	+50 bonus

BLUE TEAM SCORING

Scan detected	+30
Attacker IP blocked	+50
Webshell found	+100
Shell process killed	+150
Backdoor removed	+175
Vuln patched live	+100
SQLi in WAF logs	+60
Service restored	+80
Attack technique ID	+50
Zero exfil bonus	+300

All points awarded from REAL actions — no flags, no scripted answers

THREE DETECTION METHODS

METHOD 1 — Log Parsing

Backend reads /var/log/apache2/access.log, /var/log/auth.log, MySQL general log every 5 seconds. Regex patterns identify attack signatures and defensive actions.

METHOD 2 — Terminal Output Scanning

Every line of terminal output is scanned for patterns: "root@" after priv-esc, successful sqlmap output, successful exploit strings, iptables rule confirmations.

METHOD 3 — Admin Manual Award

Admin watches player terminal live. For ambiguous or advanced actions, admin manually clicks Award Points with custom amount and reason.

06 — ADMIN WAR ROOM

The admin dashboard gives complete visibility and control over every active player session. Admin can watch any player's terminal in real time, intervene, award or deduct points, send hints, reset the VM, or end the mission.

FEATURE	DESCRIPTION	HOW
Live Terminal Mirror	See exactly what any player is typing	Socket.io namespace stream
Session List	All active players, team, mission, score	Real-time WebSocket updates
Award Points	Manually give/remove points with reason	Admin API call → DB + broadcast
Send Hint	Push hint text to specific player's UI	Socket.io private message
Reset VM	Restore VM to clean snapshot in ~60s	Backend triggers VBoxManage CLI
Kick Player	Terminate a player SSH session	Close SSH PTY + WebSocket
Full Log	Every command player typed, with timestamps	PostgreSQL query
VM Health	CPU, RAM, disk, running services	SSH to VM + system commands
War Feed	Live narrative of all actions both teams	Event stream from scoring engine
Mission Control	Start, pause, end missions	Backend mission state machine

07 — MISSION LIFECYCLE

PHASE 1: SETUP

→ Admin selects mission scenario from dashboard

→ Backend restores VM to clean snapshot automatically

→ Vulnerable app and services start on VM

→ Monitoring tools initialize on VM

→ Both teams receive mission briefing in game UI

→ Countdown timer shown: "Mission begins in 60 seconds"

PHASE 2: ACTIVE WARFARE

→ Red team Kali terminal opens in browser

→ Blue team monitoring terminal opens in browser

→ BOTH teams now have live shells on the same VM

→ Red attacks. Blue defends. Simultaneously. In real time.

→ Every action by Red creates real events Blue must handle

→ Every defense Blue deploys creates real obstacles for Red

→ Score updates live on both teams screens every second

→ Admin war room shows full picture of the battle

PHASE 3: MISSION END

→ Timer expires OR admin manually ends mission

→ All player terminals are locked immediately

→ Final scores calculated with all bonuses applied

→ Mission report auto-generated: full timeline of both teams

→ Replay available: watch the entire battle from both sides

→ Winner announced on live leaderboard

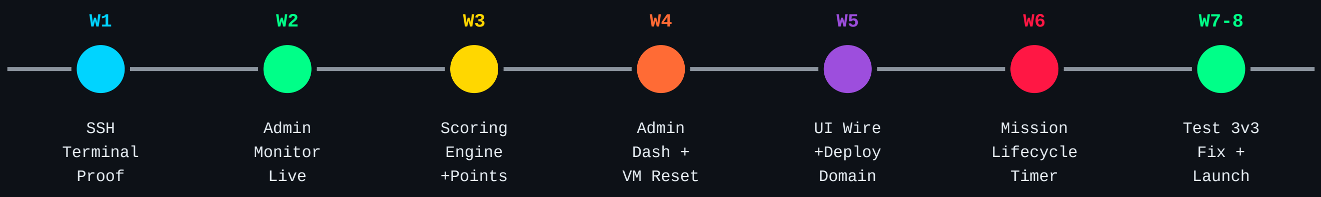
→ VM reset for next mission

08 — COMPLETE TECH STACK

COMPONENT	TECHNOLOGY	WHY THIS CHOICE
Game Frontend	React + Vite (existing)	Already built — cyberpunk UI ready
Backend API	Node.js + Express	Same runtime as SSH2 library
WebSocket	Socket.io	Real-time bidirectional events
SSH Proxy	ssh2 npm library	Node.js → VM SSH tunnel
Browser Terminal	Xterm.js	Industry standard web terminal
Database	PostgreSQL	Logs, scores, sessions, replays
Auth	JWT tokens	Stateless, scalable sessions
VM Platform	VirtualBox or VMware	On same physical server
Vulnerable App	DVWA + custom app	Real vulnerabilities, quick setup
VM Snapshots	VBoxManage CLI	Instant clean-state restore
Admin Dashboard	React page (new)	Extend existing frontend
OS (Main)	Ubuntu 22.04 LTS	Stable, good Node.js support
OS (VM)	Kali Linux (attack) + Ubuntu (target)	Real security distros
Reverse Proxy	Nginx	SSL termination + routing
Process Manager	PM2	Keep Node.js alive in production

09 — BUILD TIMELINE

Build in strict sequence. Each week proves a specific thing works before the next. Do not skip phases or build UI before the core SSH pipe is proven.



W1 — SSH Terminal Proof

W2 — Admin Monitor Live

W3 — Scoring Engine +Points

W4 — Admin Dash + VM Reset

W5 — UI Wire +Deploy Domain

W6 — Mission Lifecycle Timer

W7-8 — Test 3v3 Fix + Launch

WEEK 1	**SSH TERMINAL PROOF** Install ssh2 + Xterm.js + Socket.io. Build ssh-proxy.js. Get browser terminal connected to VM. Type whoami, see root respond. This single proof validates the entire core concept.
WEEK 2	**ADMIN MONITORING** Tee terminal output stream to admin Socket.io namespace. Admin page shows live mirror of any player terminal. Every command logged to database with timestamp.
WEEK 3	**SCORING ENGINE** Build log parser (reads VM logs every 5s). Build terminal output scanner (regex on all output). Build scoring API. Live leaderboard via Socket.io broadcast.
WEEK 4	**ADMIN DASHBOARD** Admin player list, live terminal mirror, award points button, VM reset button (triggers VBoxManage snapshot restore), kick player, send hint, war feed.
WEEK 5	**UI WIRE + DEPLOY** Connect existing React frontend (trailer → team select → agent select → mission briefing → live terminal). Deploy to live domain with Nginx + SSL.
WEEK 6	**MISSION LIFECYCLE** Mission start triggers VM snapshot restore + service init. Mission timer countdown. Mission end locks terminals + generates report. Full mission replay system.
WEEK 7-8	**TEST 3v3 + LAUNCH** Run with 6 real players, 3 per team. Break everything. Fix everything. Tune scoring weights. Polish war feed narrative. Public launch.

10 — KEY DIFFERENTIATORS vs CTF

ASPECT	CTF GAME	NEXUS PROTOCOL
Target	✗ Static challenge file	✓ Live running VM system
Objective	✗ Find hidden flag string	✓ Actually compromise/defend the system
Opponent	✗ No live opponent	✓ Blue Team actively fighting back
Terminal	✗ Fake / scripted responses	✓ Real Linux bash, real tool output
Tools	✗ Limited / sandboxed	✓ Full Kali toolkit, real commands
Outcome	✗ Predetermined answer exists	✓ No predetermined outcome
Scoring	✗ Submit correct flag = points	✓ Real technical actions = points
Environment	✗ Never changes	✓ Blue can patch, Red must adapt
Replay	✗ Not applicable	✓ Full battle timeline replay
Skill built	✗ Puzzle solving	✓ Real pentest / SOC job skills
Time	✗ Solve at your pace	✓ Live timer, real pressure

YOUR IMMEDIATE ASSIGNMENT

STEP 1 npm install ssh2 socket.io

STEP 2 Create ssh-proxy.js — one file that opens SSH to your VM

STEP 3 Add Xterm.js to your React frontend

STEP 4 Connect Xterm.js → WebSocket → ssh-proxy → VM

STEP 5 Type whoami in browser — see root respond

If that works: you have proven your entire concept.
Everything else — UI, scoring, admin, missions — is already half-built.
The SSH proxy is the only missing piece. Build it this week.