Embedding a VNC Remote Desktop inside an atvise Display

Product: atvise scada (tested on version 3.15) **Topic:** Integration / Remote access
Author: Vester Business SL **Date:** October 2026

Summary

atvise does not include a built-in VNC client. Because atvise is a fully web-based (HTML5/SVG) platform, you can still embed a browser-based VNC client directly inside an atvise display. Operators can then view and control the screen of a plant device (an HMI panel, an industrial PC, a Raspberry Pi, a virtual machine, and so on) without leaving the SCADA and without installing an external VNC viewer.

This article explains how we built and tested this integration using two open-source components:

- **noVNC**, an HTML5 VNC client that runs in the browser.
- **websockify**, a small gateway that translates WebSocket traffic from the browser into the standard VNC protocol (RFB).

The result is a VNC session rendered inside an atvise display, with full mouse and keyboard control.

> **Scope**
> The setup described in Steps 1–4 is a **proof of concept for a trusted local network**. It does not meet the requirements of **IEC 62443** on its own. The section "IEC 62443 and production use" explains the gaps and how to close them in a production installation.

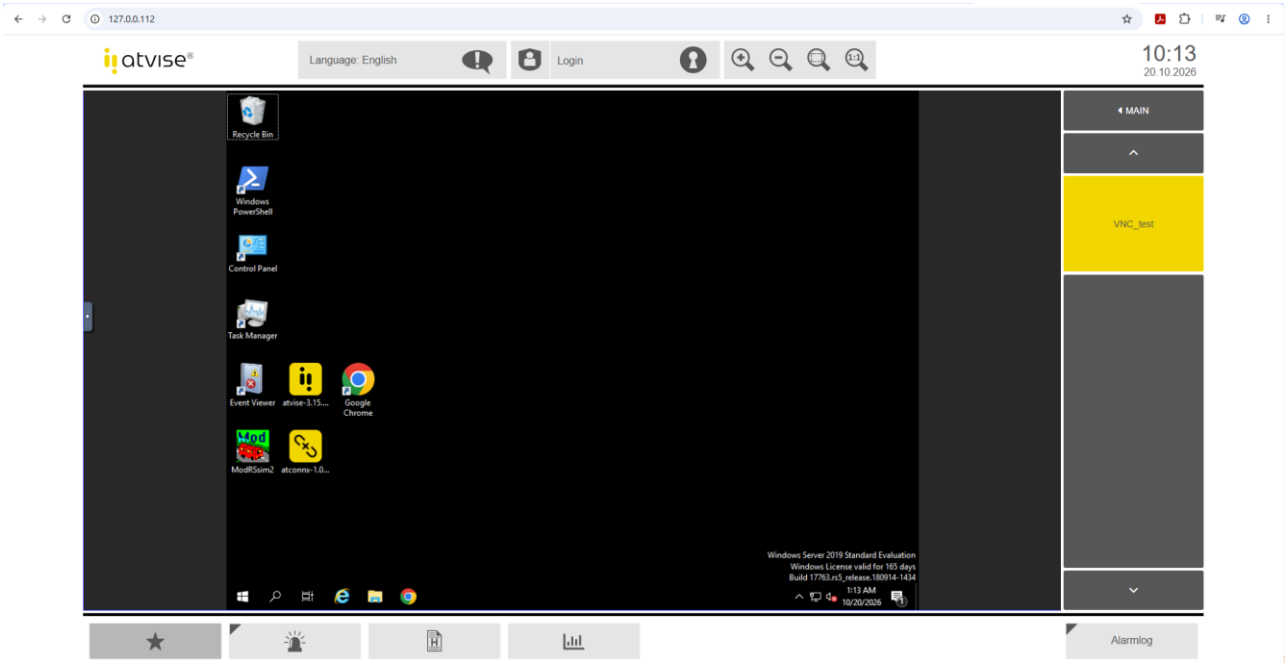


Figure 1. The final result: a remote Windows desktop running inside an atvise display.

How it works

Browsers cannot open raw TCP connections, so they cannot talk to a VNC server (port 5900) directly. websockify sits in the middle: it serves the noVNC web client and relays its WebSocket connection to the VNC server.

```
Browser (atvise web client)
  atvise display
    iframe -> noVNC
        |  HTTP + WebSocket (port 6080)
        v
Gateway: websockify + noVNC   (can run on the atvise server)
        |  VNC / RFB (port 5900)
        v
Target device (VNC server)
```

Key points:

- The **browser** connects to the gateway, not to the target device. The client PC only needs network access to the gateway on port 6080.
- The **gateway** needs network access to the target device on port 5900.
- **atvise** only hosts the display that contains the noVNC client in an iframe. No atvise server-side changes are required.

Requirements

Component	Used in our test	Notes
atvise scada	3.15	The approach only uses a standard iframe, so it should not depend on the atvise version. We have only verified it on 3.15.
Target device	Windows virtual machine	Any device that runs a standard VNC server.
VNC server	TightVNC Server 2.x	Must use standard VNC authentication (see the note on RealVNC below).
Gateway host	The atvise server itself (Windows)	Any machine that can reach both the clients and the target.
Python	3.x	Required to run websockify.
Git	Any recent version	To download noVNC. You can also download the ZIP from GitHub.

Step 1: Prepare the VNC server on the target device

In our test, the target was a Windows virtual machine reached through Remote Desktop (RDP).

1. Download TightVNC from https://www.tightvnc.com/download.php (64-bit installer) and run it on the target device.
2. Choose a **Custom** install. Only **TightVNC Server** is needed; the Viewer is not required.
3. Keep these options enabled:
   - Register TightVNC Server as a system service
   - Add exception for TightVNC to Windows Firewall
4. Set a **Password for Remote Access**. VNC authentication uses at most 8 characters. You can optionally set a **View-Only Password** too.
5. From the gateway machine, check that the VNC port is reachable. Run this in **PowerShell** (it does not work in `cmd`):

```
Test-NetConnection <TARGET_IP> -Port 5900
```

The result must show `TcpTestSucceeded : True`.

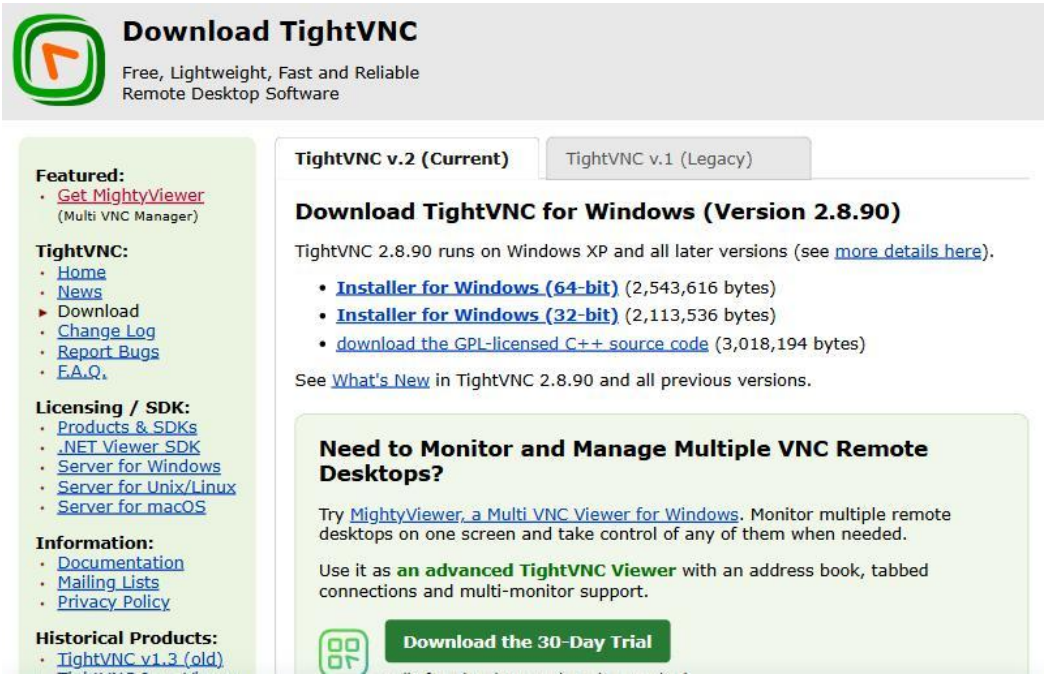


Figure 2. TightVNC download page. Use the 64-bit Windows installer.

> **Using RDP and VNC on the same Windows machine**
>
> While an RDP session is open, Windows detaches the session from the physical console. A VNC server will then show the lock screen or a black screen. Use RDP only to install TightVNC, then **close the RDP session** and log in through VNC.

> **Using RealVNC Server (for example, on Raspberry Pi OS)**
>
> By default, RealVNC uses its own authentication and encryption, which noVNC does not support. Configure it as follows:
>
> - Authentication: **VNC password** (`VncAuth`)
> - Encryption: **Prefer off** (or Always off)
>
> If encryption stays enabled, noVNC will fail to connect even when the password is correct.

Step 2: Install the gateway (noVNC + websockify)

Run these commands in PowerShell on the gateway machine. In our case, that was the atvise server.

Download noVNC. Choose a folder outside synced folders such as OneDrive:

```
git clone https://github.com/novnc/noVNC.git C:\vnc-gateway\noVNC
```

Install websockify:

```
python -m pip install websockify
```

Start the gateway. It listens on port 6080 and forwards connections to the target's VNC server:

```
python -m websockify --web C:\vnc-gateway\noVNC 6080 <TARGET_IP>:5900
```

```
[notice] To update, run: python.exe -m pip install --upgrade pip
PS C:\Users\JordiRoman> python -m websockify --web C:\vnc-gateway\noVNC 6080 192.168.3.153:5900
WARNING: no 'resource' module, daemonizing is disabled
WebSocket server settings:
  - Listen on :6080
  - Web server. Web root: C:\vnc-gateway\noVNC
  - No SSL/TLS support (no cert file)
  - proxying from :6080 to 192.168.3.153:5900
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
WARNING: no 'resource' module, daemonizing is disabled
```

Figure 3. websockify running: listening on port 6080, serving noVNC and proxying to the VNC server. The 'resource module' warnings are harmless on Windows.

- Keep this window open. Closing it stops the gateway. For a permanent installation, run websockify as a Windows service or a scheduled task.
- If Windows Firewall asks, allow port 6080 so other client PCs on the LAN can reach the gateway.

Test the gateway on its own. Before touching atvise, open `http://localhost:6080/vnc.html` in a browser, click **Connect**, and enter the VNC password. You should see and control the remote desktop. If this test fails, the atvise display will fail too, so fix this step first.



Figure 4. The noVNC client served by websockify at http://localhost:6080/vnc.html.

When the test works, close that browser tab. Some VNC servers accept only one connection at a time.

Step 3: Create the atvise display

1. In atvise builder, create a new display, for example `VNC_test`.
2. Draw a **rectangle** in the area where the remote screen should appear. Set its **id** to `vnc_area`.
3. Add the following display script:

```
// Placeholder rectangle drawn in atvise builder
var area = document.getElementById("vnc_area");

// IMPORTANT: use the real DOM document (see "Pitfall" below)
var doc = area.ownerDocument;

// SVG container for HTML content, same position and size as the rectangle
var fo = doc.createElementNS("http://www.w3.org/2000/svg", "foreignObject");
["x", "y", "width", "height", "transform"].forEach(function (a) {
    if (area.getAttribute(a)) fo.setAttribute(a, area.getAttribute(a));
});

// noVNC client served by websockify
var iframe = doc.createElementNS("http://www.w3.org/1999/xhtml", "iframe");
iframe.setAttribute("src", "http://" + location.hostname +
    ":6080/vnc.html?autoconnect=1&resize=scale&reconnect=1&view_only=0");
iframe.setAttribute("style", "width:100%;height:100%;border:0");

fo.appendChild(iframe);
area.parentNode.appendChild(fo);
```

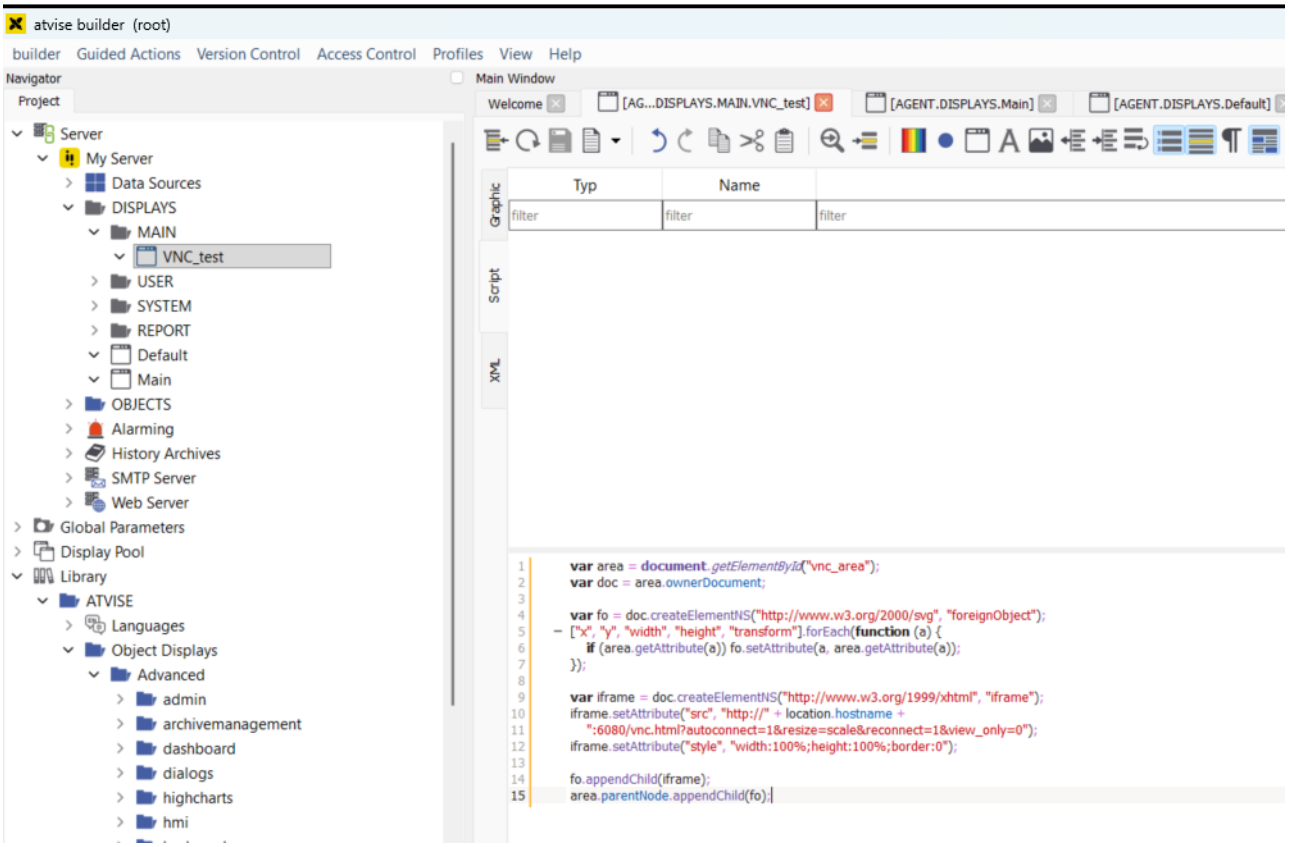


Figure 5. The VNC_test display and its script in atvise builder.

4. Open the display from your navigation, or from a button with:

```
webMI.display.openDisplay("AGENT.DISPLAYS.VNC_test");
```

Adjust the node path to where the display sits in your project.

Pitfall: "Illegal invocation" when creating elements

Our first version of the script called `document.createElementNS(...)`. The display only showed the empty rectangle, and the browser console reported:

```
Uncaught TypeError: Illegal invocation
```

Inside atvise display scripts, `document` is a wrapper object provided by `webMI.proxy`. It supports methods such as `getElementById`, but calling `createElementNS` on it fails. The fix is to get the real DOM document from an existing element:

```
var doc = area.ownerDocument;
doc.createElementNS(...);
```

noVNC URL parameters

Parameter	Value used	Meaning
`autoconnect`	`1`	Connect as soon as the display opens.
`resize`	`scale`	Scale the remote screen to fit the rectangle. Use `remote` to request a resolution change on the server instead.
`reconnect`	`1`	Reconnect automatically if the connection drops.
`view_only`	`0` / `1`	`1` disables mouse and keyboard input.

Do **not** put the VNC password in the URL. noVNC prompts for it when connecting.

`location.hostname` makes the display work from any client PC, because the gateway address follows the address used to open atvise. This assumes the gateway runs on the atvise server. Otherwise, replace it with the gateway's address.

Step 4: Test

Open atvise in the browser with **http** (for example, `http://<atvise-server>`), open the display, and enter the VNC password when prompted.

Checklist:

- ☐ The remote desktop appears inside the rectangle and scales to fit.
- ☐ Mouse and keyboard work. Click inside the viewer first to give it keyboard focus.
- ☐ It works from another PC on the LAN.
- ☐ After restarting the VNC server, the viewer reconnects automatically.
- ☐ With `view_only=1`, the screen is visible but cannot be controlled.

Result: in our test on atvise 3.15, the remote desktop was fully usable from inside the display, with mouse and keyboard control (Figure 1).

Troubleshooting

Symptom	Likely cause	Fix
Only the rectangle is visible; console shows `Illegal invocation`	`createElementNS` was called on the webMI `document` wrapper	Use `area.ownerDocument.createElementNS`
Viewer area is blank or blocked when atvise runs over HTTPS	Mixed content: an HTTPS page cannot load an HTTP or WS iframe	Use http for testing. For production, serve the gateway over HTTPS/WSS (websockify `--cert` / `--key`) and use `https://` in the iframe URL.
"Failed to connect to server"	Gateway not running, port 6080 blocked, or target unreachable	Check the websockify window. Run `Test-NetConnection <TARGET_IP> -Port 5900` from the gateway.

Symptom	Likely cause	Fix
Connects, then disconnects with RealVNC	RealVNC encryption or authentication not supported by noVNC	Set authentication to VNC password and encryption to Prefer off
Black screen or lock screen (Windows target)	An RDP session is open on the same machine	Close the RDP session and log in through VNC
`winget` not recognized	winget is not available on Windows Server or some Windows 10 builds	Download the TightVNC installer manually
`Test-NetConnection` not recognized	Command run in `cmd`	Run it in PowerShell

Security considerations

- **Unencrypted traffic.** Standard VNC and the plain websockify setup do not encrypt traffic. Use them only on a trusted LAN. For anything beyond that, enable TLS on websockify (WSS) and serve atvise over HTTPS.
- **Restrict the VNC server.** Configure the target's firewall so that port 5900 accepts connections only from the gateway. Users then cannot bypass atvise and connect directly.
- **view_only is not access control.** This parameter is applied in the browser. A user who edits the URL can remove it. For real role separation (operators with full control, other users view-only), use one of these options:
  - Show the VNC display only to authorised users with atvise display permissions. Also, if the device supports it, use a separate view-only password on the VNC server: TightVNC supports this.
  - Use a gateway that enforces permissions on the server side, such as **Apache Guacamole** (see below).
- **Passwords.** VNC authentication passwords are at most 8 characters long. Treat them as a second barrier, not as the main protection.

IEC 62443 and production use

IEC 62443 is the international series of standards for the security of industrial automation and control systems. It applies to **systems, components and processes**, not to a single feature. Compliance must therefore be assessed for the complete installation, starting from a risk assessment (IEC 62443-3-2) that sets a target **security level (SL)** for each zone.

The proof of concept in this article is useful to validate the technique. It does **not** meet the system requirements of IEC 62443-3-3, because it was built for a trusted LAN test without additional security measures.

Gap analysis against IEC 62443-3-3

Foundational requirement	Proof of concept	Gap	Production measure
FR1 – Identification and authentication control	One shared VNC password (max. 8 characters)	No individual user identification (SR 1.1); weak password-based authentication (SR 1.5, SR 1.7)	Gateway with individual accounts linked to the corporate directory (AD/LDAP); multi-factor authentication where the target SL requires it

Foundational requirement	Proof of concept	Gap	Production measure
FR2 – Use control	view_only set in the browser URL	Authorisation not enforced on the server (SR 2.1); no auditable events (SR 2.8)	Roles (full control / view-only) enforced by the gateway; session events logged
FR3 – System integrity	Plain VNC and plain WebSocket	No communication integrity (SR 3.1)	TLS between browser and gateway (HTTPS/WSS); target link in an isolated network or tunnel
FR4 – Data confidentiality	Traffic in clear text	No confidentiality in transit (SR 4.1)	Same as FR3
FR5 – Restricted data flow	Gateway on the atvise server; targets accept VNC from any host	No zones or conduits defined (SR 5.1, SR 5.2)	Gateway in its own zone; target firewalls accept port 5900 only from the gateway; conduit documented
FR6 – Timely response to events	No session logging	Audit logs not available (SR 6.1)	Central logging of who connected, when and to which device; optional session recording
FR7 – Resource availability	Single gateway process	Not assessed	Size and make redundant according to the risk assessment; run the gateway as a supervised service

Recommended production architecture

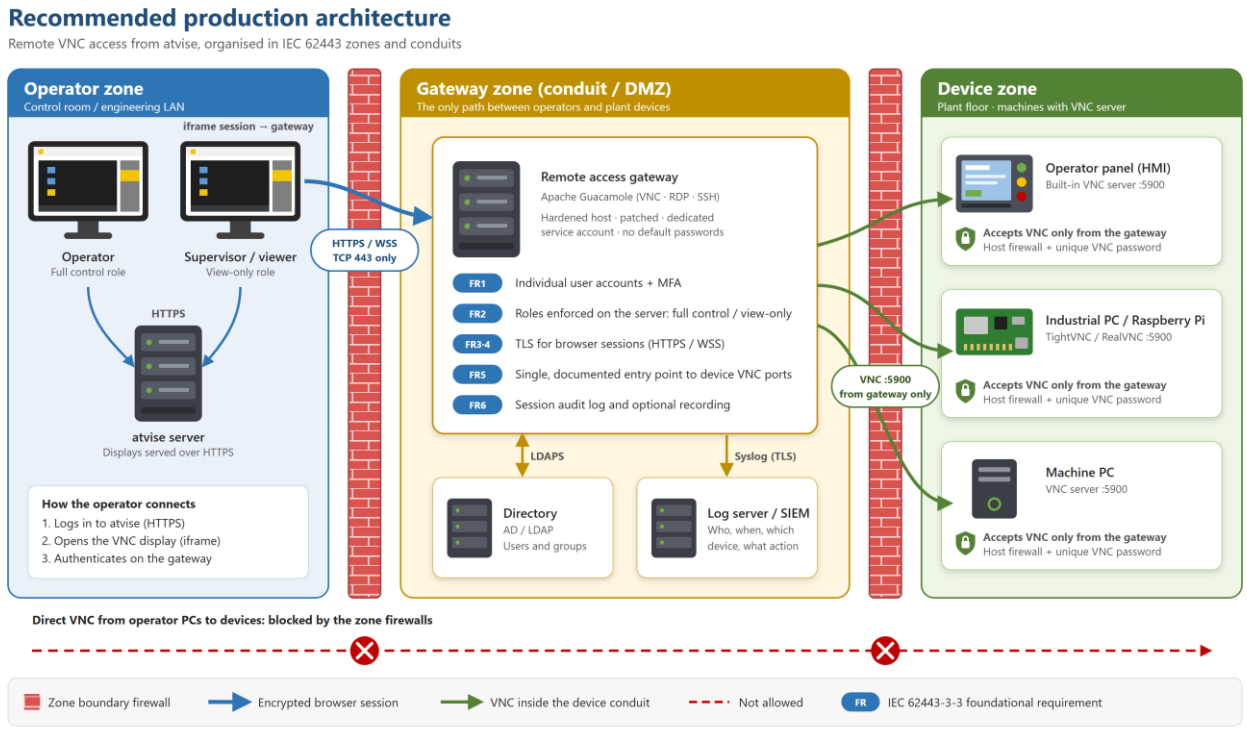


Figure 6. Recommended production architecture: operator, gateway and device zones separated by firewalls, with the gateway as the only conduit to the devices.

- **Gateway:** Apache Guacamole instead of plain websockify. It provides individual accounts, server-side read-only connections, connection management and session recording.

- **Encryption:** atvise and the gateway served over HTTPS/WSS with valid certificates. Avoid mixed HTTP/HTTPS.
- **Segmentation:** the gateway is the only path to the devices' VNC ports. Devices must not accept VNC from operator PCs directly.
- **Hardening:** dedicated service account, no default passwords, unused services disabled, patch management for the gateway OS and software.
- **Process:** include the gateway in the asset inventory, the risk assessment and the change-management procedures of the plant (IEC 62443-2-1). Ask integrators and suppliers to follow IEC 62443-2-4.

> **Disclaimer**
> This article describes a technical integration. It is not a compliance statement. Whether an installation meets IEC 62443 depends on its risk assessment, target security levels, architecture and operating procedures, and must be evaluated for each project.

Scaling up: several devices and user roles

Several devices with one gateway. websockify can route connections to different targets using a token file:

```
# tokens.cfg
press01: 192.168.10.21:5900
packer02: 192.168.10.22:5900
```

```
python -m websockify --web C:\vnc-gateway\noVNC --token-plugin TokenFile --token-source C:\vnc-gateway\tokens.cfg 6080
```

The iframe URL then selects the device:

```
http://<gateway>:6080/vnc.html?autoconnect=1&path=websockify%3Ftoken%3Dpress01
```

You can reuse a single atvise display for every device by passing the token as a display parameter.

User roles enforced on the server. For larger installations with many devices, different permission levels or session auditing, **Apache Guacamole** is a better gateway. It supports VNC, RDP and SSH. It enforces read-only connections on the server and manages users and connections centrally. It can be embedded in an atvise display with the same iframe technique.

Conclusion

atvise has no native VNC feature, but its web architecture makes the integration simple. You need an open-source gateway (websockify + noVNC), a display with an iframe, and about fifteen lines of display script. The only atvise-specific detail is creating the DOM elements through `ownerDocument` instead of the webMI `document` wrapper.

For production use, treat the proof of concept as a starting point: apply the measures in the IEC 62443 section (individual authentication, server-side roles, encryption, segmentation and audit logging), based on the risk assessment of your installation.