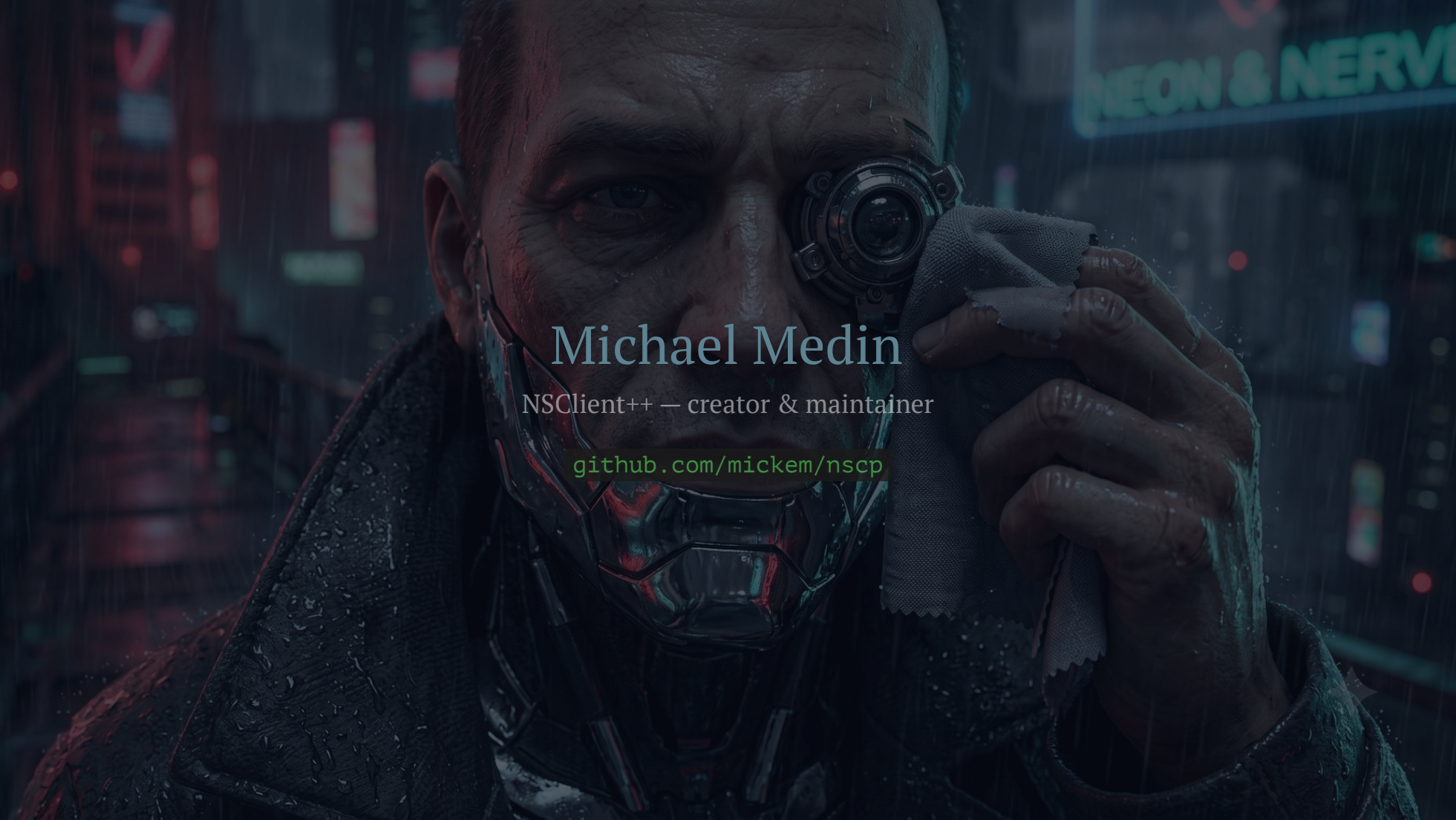
WEB-PRD-001 WEB-PRD-002 WEB-PRD-003 WEB-PRD-004 WEB-PRD-005 WEB-PRD-006 WEB-PRD-007 WEB-PRD-008 WEB-PRD-009 WEB-PRD-010 WEB-PRD-011 WEB-PRD-012 WEB-STG-001 WEB-STG-002 WEB-STG-003 WEB-STG-004 WEB-DMZ-01 WEB-DMZ-02 WEB-DMZ-03 WEB-DMZ-04 SQL-PRD-001 SQL-PRD-002 SQL-PRD-003 SQL-PRD-004 SQL-PRD-005 SQL-PRD-006 SQL-PRD-007 SQL-PRD-008 SQL-STG-001 SQL-STG-002 SQL-RPT-01 SQL-RPT-02 APP-PRD-001 APP-PRD-002 APP-PRD-003 APP-PRD-004 APP-PRD-005 APP-PRD-006 APP-PRD-007 APP-PRD-008 APP-PRD-009 APP-PRD-010 APP-PRD-011 APP-PRD-012 APP-PRD-013 APP-PRD-014 APP-PRD-015 APP-PRD-016 APP-STG-001 APP-STG-002 APP-STG-003 APP-STG-004 APP-STG-005 APP-STG-006 APP-STG-007 APP-STG-008 APP-STG-009 APP-STG-010 APP-STG-011 APP-STG-012 APP-STG-013 APP-STG-014 DC-01 DC-02 DC-03 DC-04 FS-PRD-001 FS-PRD-002 FS-PRD-003 FS-PRD-004 FS-PRD-005 FS-PRD-006 EXCH-PRD-01 EXCH-PRD-02 EXCH-PRD-03 EXCH-PRD-04 IIS-PRD-001 IIS-PRD-002 IIS-PRD-003 IIS-PRD-004 IIS-PRD-005 IIS-PRD-006 RDS-PRD-01 RDS-PRD-02 RDS-PRD-03 RDS-PRD-04 RDS-PRD-05 RDS-PRD-06 HV-PRD-01 HV-PRD-02 HV-PRD-03 HV-PRD-04 HV-PRD-05 HV-PRD-06 HV-PRD-07 HV-PRD-08 CTX-PRD-01 CTX-PRD-02 CTX-PRD-03 CTX-PRD-04 BACKUP-01 BACKUP-02 BACKUP-03 WSUS-01 SCCM-01 MON-01 AD-SYNC-01 VPN-GW-01 VPN-GW-02 POS-STORE-001 POS-STORE-002 POS-STORE-003 POS-STORE-004 POS-STORE-005 POS-STORE-006 POS-STORE-007 POS-STORE-008 POS-STORE-009 POS-STORE-010 POS-STORE-011 POS-STORE-012 KIOSK-001 KIOSK-002 KIOSK-003 KIOSK-004 KIOSK-005 KIOSK-006 KIOSK-007 KIOSK-008 LAB-DEV-001 LAB-DEV-002 LAB-DEV-003 LAB-DEV-004 LAB-DEV-005 LAB-DEV-006 TS-LEGACY-01 TS-LEGACY-02 TS-LEGACY-03 FILE-ARCH-01 FILE-ARCH-02 PRINT-01 PRINT-02 PRINT-03 SQL-DEV-001 SQL-DEV-002 SQL-DEV-003 WEB-PRD-013 WEB-



Michael Medin

NSClient++ — creator & maintainer

`github.com/mickem/nscp`

Builder

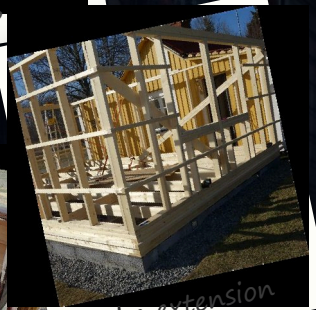
I love to build things!

Software , Legos, Wood...

1983

balancing robot

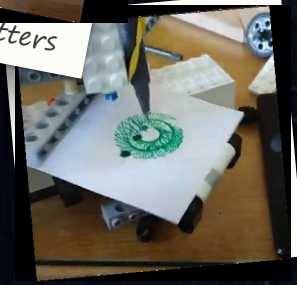
still 1983



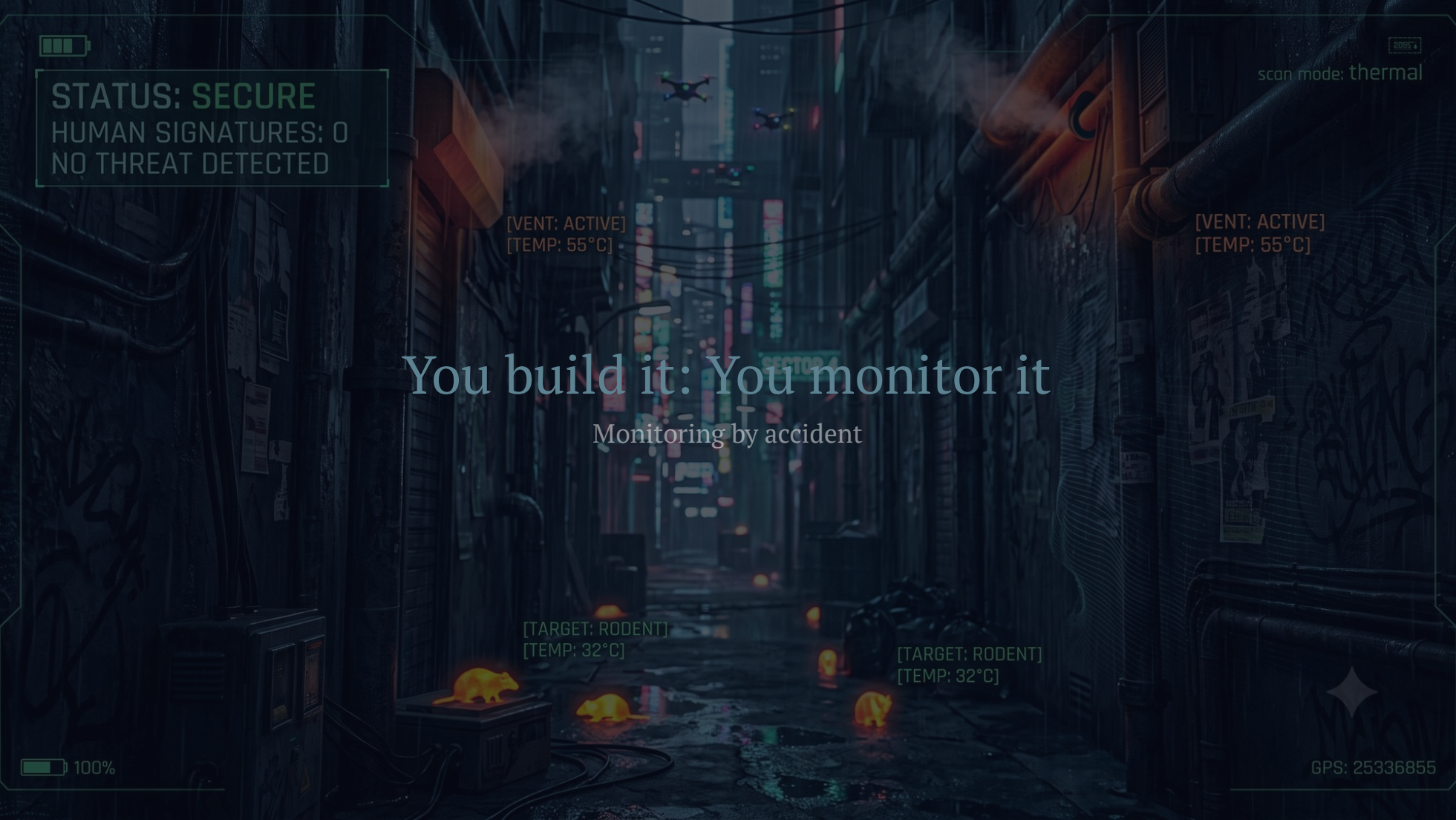
the extension

treasure chest

window shutters



lego plotter



You build it: You monitor it

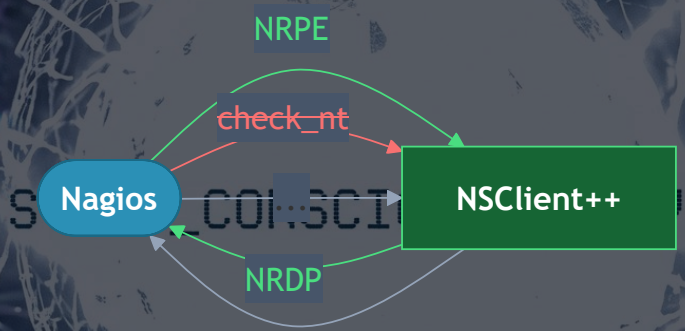
Monitoring by accident

What is NSClient++?

NSClient++ — a monitoring agent for Windows and Linux.

- Runs as a service
- Answers questions about the box
- Or pushes results out on its own
- Open source — Apache-2.0 OR GPL-2.0-only

How it talks to Nagios



We wont touch this today!

What it checks out of the box

Command	Checks	Default warn	Default crit
`check_cpu`	CPU load	> 80%	> 90%
`check_memory`	Memory	> 79%	> 89%
`check_drivesize`	Disk space	> 79% used	> 89% used
`check_service`	Windows services	auto-start, stopped	—
`check_eventlog`	Event log	warnings	errors / critical
`check_uptime`	Time since reboot	< 2d	< 1d

And when the built-ins aren’t enough

External scripts

```
[/settings/external scripts/scripts]
check_my_app = scripts\check_my_app.ps1
```

Batch · PowerShell · VBScript · Lua · Python

Exit code: `0` OK · `1` WARNING · `2` CRITICAL · `3` UNKNOWN

Output: `Status text|'metric'=value;warn;crit;min;max`

What’s new: built-in checks

18 → 93

check_mssql
check_mssql_backup
check_mssql_databases
check_mssql_jobs
check_mssql_query
check_mysql
check_mysql_query
check_share
check_uncpath
check_disk_write
check_shadowcopy
check_storagepool
check_disk_health
check_disk_io
check_mount
check_single_file
check_defender
check_bitlocker
check_secureboot
check_antivirus
check_certificate
check_firewall
check_firewall_rules
check_nla
check_local_accounts
check_group_members
check_users
check_activation
check_file_security
check_tcp
check_http
check_dns
check_ssh
check_ntp_offset
check_connection
check_nsclient_web_online
check_apache_status
check_nginx_status
check_phpfpm_status
check_tomcat_status
check_iis_sites
check_iis_app_pools
check_iis_worker_processes
check_iis_request_queues
check_rds_sessions
check_rds_session_load
check_rds_broker
check_rds_licenses
check_os_updates
check_pending_reboot
check_patch_age
check_printqueue
check_printjobs
check_registry_key
check_registry_value
check_installed_software
check_hostname
check_w32time
check_load
check_cpu_utilization
check_cpu_frequency
check_kernel_stats
check_kernel_memory
check_swap_io
check_temperature
check_battery
check_hardware
check_process_history
check_process_history_new
check_docker_df
check_docker_info
check_docker_restarts
check_docker_stats
check_nscp_update

WHAT'S NEW — WHAT THAT ADDS UP TO

7 Databases

SQL Server · MySQL

check_mssql
check_mssql_backup
check_mssql_databases
check_mssql_jobs
check_mssql_query
check_mysql
check_mysql_query
check_share
check_uncpath
check_disk_write
check_shadowcopy
check_storagepool
check_disk_health
check_disk_io
check_mount
check_single_file
check_defender
check_bitlocker
check_secureboot
check_antivirus
check_certificate
check_firewall
check_firewall_rules
check_nla
check_local_accounts
check_group_members
check_users
check_activation
check_file_security
check_tcp
check_http
check_dns
check_ssh
check_ntp_offset
check_connections
check_nsclient_web_online
check_apache_status
check_nginx_status
check_phpfpm_status
check_tomcat_status
check_iis_sites
check_iis_app_pools
check_iis_worker_processes
check_iis_request_queues
check_rds_sessions
check_rds_session_load
check_rds_broker
check_rds_licenses
check_os_updates
check_pending_reboot
check_patch_age
check_printqueue
check_printjobs
check_registry_key
check_registry_value
check_installed_software
check_hostname
check_w32time
check_load
check_cpu_utilization
check_cpu_frequency
check_kernel_stats
check_kernel_memory
check_swap_io
check_temperature
check_battery
check_hardware
check_process_history
check_process_history_new
check_docker_df
check_docker_info
check_docker_restarts
check_docker_stats
check_nscp_update

WHAT'S NEW — WHAT THAT ADDS UP TO

9 File servers & disk

shares, write probes, disk health

check_mssql
check_mssql_backup
check_mssql_databases
check_mssql_jobs
check_mssql_query
check_mysql
check_mysql_query
check_share
check_uncpath
check_disk_write
check_shadowcopy
check_storagepool
check_disk_health
check_disk_io
check_mount
check_single_file
check_defender
check_bitlocker
check_secureboot
check_antivirus
check_certificate
check_firewall
check_firewall_rules
check_nla
check_local_accounts
check_group_members
check_users
check_activation
check_file_security
check_tcp
check_http
check_dns
check_ssh
check_ntp_offset
check_connections
check_nsclient_web_online
check_apache_status
check_nginx_status
check_phpfpm_status
check_tomcat_status
check_iis_sites
check_iis_app_pools
check_iis_worker_processes
check_iis_request_queues
check_rds_sessions
check_rds_session_load
check_rds_broker
check_rds_licenses
check_os_updates
check_pending_reboot
check_patch_age
check_printqueue
check_printjobs
check_registry_key
check_registry_value
check_installed_software
check_hostname
check_w32time
check_load
check_cpu_utilization
check_cpu_frequency
check_kernel_stats
check_kernel_memory
check_swap_io
check_temperature
check_battery
check_hardware
check_process_history
check_process_history_new
check_docker_df
check_docker_info
check_docker_restarts
check_docker_stats
check_nscp_update

WHAT'S NEW — WHAT THAT ADDS UP TO

13 Security posture

Defender, BitLocker, certificates, firewall

check_mssql
check_mssql_backup
check_mssql_databases
check_mssql_jobs
check_mssql_query
check_mysql
check_mysql_query
check_share
check_uncpath
check_disk_write
check_shadowcopy
check_storagepool
check_disk_health
check_disk_io
check_mount
check_single_file
check_defender
check_bitlocker
check_secureboot
check_antivirus
check_certificate
check_firewall
check_firewall_rules
check_nla
check_local_accounts
check_group_members
check_users
check_activation
check_file_security
check_tcp
check_http
check_dns
check_ssh
check_ntp_offset
check_connections
check_nsclient_web_online
check_apache_status
check_nginx_status
check_phpfpm_status
check_tomcat_status
check_iis_sites
check_iis_app_pools
check_iis_worker_processes
check_iis_request_queues
check_rds_sessions
check_rds_session_load
check_rds_broker
check_rds_licenses
check_os_updates
check_pending_reboot
check_patch_age
check_printqueue
check_printjobs
check_registry_key
check_registry_value
check_installed_software
check_hostname
check_w32time
check_load
check_cpu_utilization
check_cpu_frequency
check_kernel_stats
check_kernel_memory
check_swap_io
check_temperature
check_battery
check_hardware
check_process_history
check_process_history_new
check_docker_df
check_docker_info
check_docker_restarts
check_docker_stats
check_nscp_update

WHAT'S NEW — WHAT THAT ADDS UP TO

11 Network, from the box

ports, HTTP, DNS — plus your web tier

check_mssql
check_mssql_backup
check_mssql_databases
check_mssql_jobs
check_mssql_query
check_mysql
check_mysql_query
check_share
check_uncpath
check_disk_write
check_shadowcopy
check_storagepool
check_disk_health
check_disk_io
check_mount
check_single_file
check_defender
check_bitlocker
check_secureboot
check_antivirus
check_certificate
check_firewall
check_firewall_rules
check_nla
check_local_accounts
check_group_members
check_users
check_activation
check_file_security
check_tcp
check_http
check_dns
check_ssh
check_ntp_offset
check_connections
check_nsclient_web_online
check_apache_status
check_nginx_status
check_phpfpm_status
check_tomcat_status
check_iis_sites
check_iis_app_pools
check_iis_worker_processes
check_iis_request_queues
check_rds_sessions
check_rds_session_load
check_rds_broker
check_rds_licenses
check_os_updates
check_pending_reboot
check_patch_age
check_printqueue
check_printjobs
check_registry_key
check_registry_value
check_installed_software
check_hostname
check_w32time
check_load
check_cpu_utilization
check_cpu_frequency
check_kernel_stats
check_kernel_memory
check_swap_io
check_temperature
check_battery
check_hardware
check_process_history
check_process_history_new
check_docker_df
check_docker_info
check_docker_restarts
check_docker_stats
check_nscp_update

WHAT'S NEW — WHAT THAT ADDS UP TO

8 IIS & Remote Desktop

sites, app pools · sessions, licensing

check_mssql
check_mssql_backup
check_mssql_databases
check_mssql_jobs
check_mssql_query
check_mysql
check_mysql_query
check_share
check_uncpath
check_disk_write
check_shadowcopy
check_storagepool
check_disk_health
check_disk_io
check_mount
check_single_file
check_defender
check_bitlocker
check_secureboot
check_antivirus
check_certificate
check_firewall
check_firewall_rules
check_nla
check_local_accounts
check_group_members
check_users
check_activation
check_file_security
check_tcp
check_http
check_dns
check_ssh
check_ntp_offset
check_connections
check_nsclient_web_online
check_apache_status
check_nginx_status
check_phpfpm_status
check_tomcat_status
check_iis_sites
check_iis_app_pools
check_iis_worker_processes
check_iis_request_queues
check_rds_sessions
check_rds_session_load
check_rds_broker
check_rds_licenses
check_os_updates
check_pending_reboot
check_patch_age
check_printqueue
check_printjobs
check_registry_key
check_registry_value
check_installed_software
check_hostname
check_w32time
check_load
check_cpu_utilization
check_cpu_frequency
check_kernel_stats
check_kernel_memory
check_swap_io
check_temperature
check_battery
check_hardware
check_process_history
check_process_history_new
check_docker_df
check_docker_info
check_docker_restarts
check_docker_stats
check_nscp_update

WHAT'S NEW — WHAT THAT ADDS UP TO

10 Windows operations

patches, print, registry, time sync

check_mssql
check_mssql_backup
check_mssql_databases
check_mssql_jobs
check_mssql_query
check_mysql
check_mysql_query
check_share
check_uncpath
check_disk_write
check_shadowcopy
check_storagepool
check_disk_health
check_disk_io
check_mount
check_single_file
check_defender
check_bitlocker
check_secureboot
check_antivirus
check_certificate
check_firewall
check_firewall_rules
check_nla
check_local_accounts
check_group_members
check_users
check_activation
check_file_security
check_tcp
check_http
check_dns
check_ssh
check_ntp_offset
check_connections
check_nsclient_web_online
check_apache_status
check_nginx_status
check_phpfpm_status
check_tomcat_status
check_iis_sites
check_iis_app_pools
check_iis_worker_processes
check_iis_request_queues
check_rds_sessions
check_rds_session_load
check_rds_broker
check_rds_licenses
check_os_updates
check_pending_reboot
check_patch_age
check_printqueue
check_printjobs
check_registry_key
check_registry_value
check_installed_software
check_hostname
check_w32time
check_load
check_cpu_utilization
check_cpu_frequency
check_kernel_stats
check_kernel_memory
check_swap_io
check_temperature
check_battery
check_hardware
check_process_history
check_process_history_new
check_docker_df
check_docker_info
check_docker_restarts
check_docker_stats
check_nscp_update

WHAT'S NEW — WHAT THAT ADDS UP TO

11 The system, deeper

load, kernel, temperature, hardware

check_mssql
check_mssql_backup
check_mssql_databases
check_mssql_jobs
check_mssql_query
check_mysql
check_mysql_query
check_share
check_uncpath
check_disk_write
check_shadowcopy
check_storagepool
check_disk_health
check_disk_io
check_mount
check_single_file
check_defender
check_bitlocker
check_secureboot
check_antivirus
check_certificate
check_firewall
check_firewall_rules
check_nla
check_local_accounts
check_group_members
check_users
check_activation
check_file_security
check_tcp
check_http
check_dns
check_ssh
check_ntp_offset
check_connections
check_nsclient_web_online
check_apache_status
check_nginx_status
check_phpfpm_status
check_tomcat_status
check_iis_sites
check_iis_app_pools
check_iis_worker_processes
check_iis_request_queues
check_rds_sessions
check_rds_session_load
check_rds_broker
check_rds_licenses
check_os_updates
check_pending_reboot
check_patch_age
check_printqueue
check_printjobs
check_registry_key
check_registry_value
check_installed_software
check_hostname
check_w32time
check_load
check_cpu_utilization
check_cpu_frequency
check_kernel_stats
check_kernel_memory
check_swap_io
check_temperature
check_battery
check_hardware
check_process_history
check_process_history_new
check_docker_df
check_docker_info
check_docker_restarts
check_docker_stats
check_nscp_update

WHAT'S NEW — WHAT THAT ADDS UP TO

5 Containers & self

Docker · the agent's own updates

check_mssql
check_mssql_backup
check_mssql_databases
check_mssql_jobs
check_mssql_query
check_mysql
check_mysql_query
check_share
check_uncpath
check_disk_write
check_shadowcopy
check_storagepool
check_disk_health
check_disk_io
check_mount
check_single_file
check_defender
check_bitlocker
check_secureboot
check_antivirus
check_certificate
check_firewall
check_firewall_rules
check_nla
check_local_accounts
check_group_members
check_users
check_activation
check_file_security
check_tcp
check_http
check_dns
check_ssh
check_ntp_offset
check_connections
check_nsclient_web_online
check_apache_status
check_nginx_status
check_phpfpm_status
check_tomcat_status
check_iis_sites
check_iis_app_pools
check_iis_worker_processes
check_iis_request_queues
check_rds_sessions
check_rds_session_load
check_rds_broker
check_rds_licenses
check_os_updates
check_pending_reboot
check_patch_age
check_printqueue
check_printjobs
check_registry_key
check_registry_value
check_installed_software
check_hostname
check_w32time
check_load
check_cpu_utilization
check_cpu_frequency
check_kernel_stats
check_kernel_memory
check_swap_io
check_temperature
check_battery
check_hardware
check_process_history
check_process_history_new
check_docker_df
check_docker_info
check_docker_restarts
check_docker_stats
check_nscp_update

What’s new: locked down by default

THE AGENT GOT STRICTER ABOUT ITSELF

empty `allowed hosts` now means nobody

it used to mean everybody

What’s new: locked down by default

THE AGENT GOT STRICTER ABOUT ITSELF

trust comes from the Windows CA store

the machine's own `ROOT` store, so an internal CA you already trust just works `0.12.3`

What’s new: locked down by default

THE AGENT GOT STRICTER ABOUT ITSELF

certificates are verified by default

For almost everything

What’s new: locked down by default

THE AGENT GOT STRICTER ABOUT ITSELF

a real internal permission model

per-command allow-lists, roles, client-certificate identities

What’s new: locked down by default

THE AGENT GOT STRICTER ABOUT ITSELF

a modern Windows file layout

config and state out of `%ProgramFiles%` , ACL'd to SYSTEM + Administrators

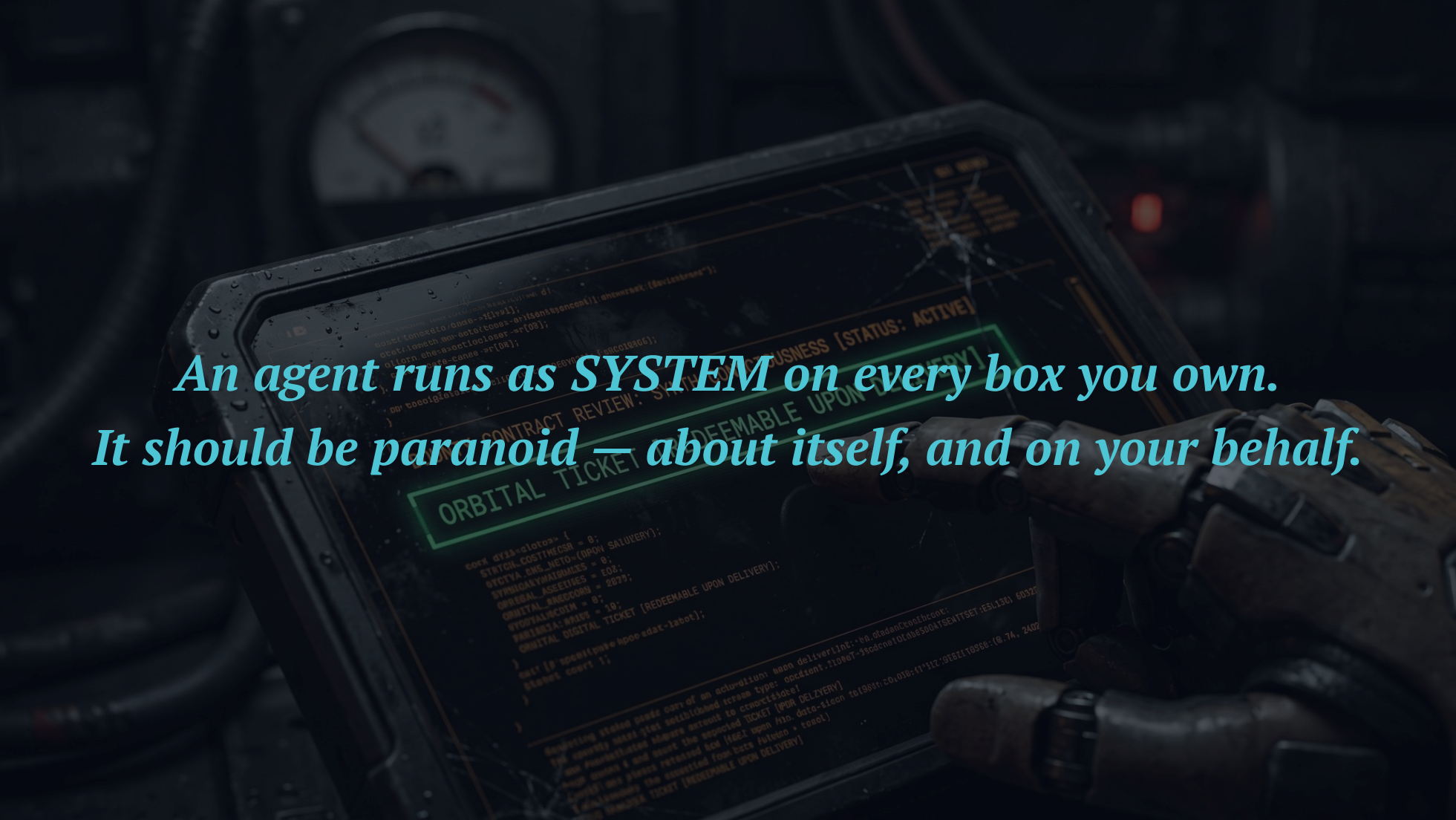
`currently opt-in: LAYOUT=modern`

What’s new: locked down by default

AND IT LEARNED TO CHECK ***YOURS***

13 of the new checks are security checks

- check_defender
- check_antivirus
- check_bitlocker
- check_secureboot
- check_firewall
- check_firewall_rules
- check_certificate
- check_nla
- check_activation
- check_file_security
- check_local_accounts
- check_group_members
- check_users



An agent runs as SYSTEM on every box you own.
It should be paranoid — about itself, and on your behalf.

Linux

experimental → near-parity

What’s new: everything else

- **Install from your package manager**: winget · Chocolatey · Scoop
- **More places to send results** — Checkmk · Icinga 2’s REST API · NSCA-ng
- **License**: now `Apache-2.0` OR `GPL-2.0-only` — pick whichever your lawyers like
  - Mongoose require: `GPL-2.0-only` (slowly migrating to beast)
- **Refreshed modules**: All(?) module have gotten a refresh where bugs have been fixed and documentation updated
- **Tests**: Test coverage is now well above 50% with 100% on track
- **Bug fixes**: Many many bugs were squshed (and new ones likely created)
- **Interactive changelog fixes**: Allows you to filter on version and if it requires an action

Six new modules. Seventy-four new checks. One year.

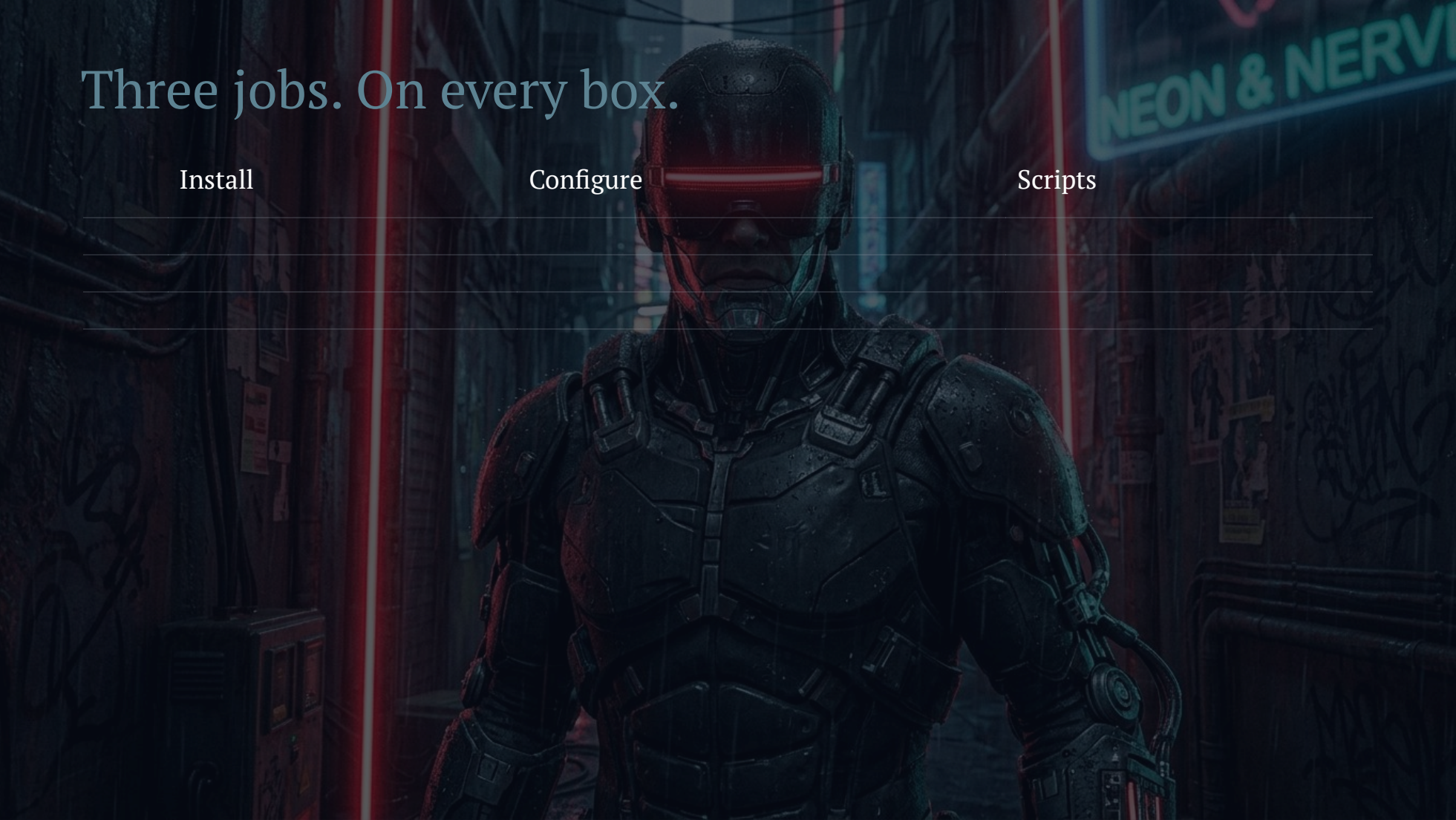
… and a big Security focus!

Now: Back to the regular feature

WEB-PRD-001 WEB-PRD-002 WEB-PRD-003 WEB-PRD-004 WEB-PRD-005 WEB-PRD-006 WEB-PRD-007 WEB-PRD-008 WEB-PRD-009 WEB-PRD-010 WEB-PRD-011 WEB-PRD-012 WEB-STG-001 WEB-STG-002 WEB-STG-003 WEB-STG-004 WEB-DMZ-01 WEB-DMZ-02 WEB-DMZ-03 WEB-DMZ-04 SQL-PRD-001 SQL-PRD-002 SQL-PRD-003 SQL-PRD-004 SQL-PRD-005 SQL-PRD-006 SQL-PRD-007 SQL-PRD-008 SQL-STG-001 SQL-STG-002 SQL-RPT-01 SQL-RPT-02 APP-PRD-001 APP-PRD-002 APP-PRD-003 APP-PRD-004 APP-PRD-005 APP-PRD-006 APP-PRD-007 APP-PRD-008 APP-PRD-009 APP-PRD-010 APP-PRD-011 APP-PRD-012 APP-PRD-013 APP-PRD-014 APP-PRD-015 APP-PRD-016 APP-STG-001 APP-STG-002 APP-STG-003 APP-STG-004 APP-STG-005 APP-STG-006 APP-STG-007 APP-STG-008 APP-STG-009 APP-STG-010 APP-STG-011 APP-STG-012 APP-STG-013 APP-STG-014 DC-01 DC-02 DC-03 DC-04 FS-PRD-001 FS-PRD-002 FS-PRD-003 FS-PRD-004 FS-PRD-005 FS-PRD-006 EXCH-PRD-01 EXCH-PRD-02 EXCH-PRD-03 EXCH-PRD-04 IIS-PRD-001 IIS-PRD-002 IIS-PRD-003 IIS-PRD-004 IIS-PRD-005 IIS-PRD-006 RDS-PRD-01 RDS-PRD-02 RDS-PRD-03 RDS-PRD-04 RDS-PRD-05 RDS-PRD-06 HV-PRD-01 HV-PRD-02 HV-PRD-03 HV-PRD-04 HV-PRD-05 HV-PRD-06 HV-PRD-07 HV-PRD-08 CTX-PRD-01 CTX-PRD-02 CTX-PRD-03 CTX-PRD-04 BACKUP-01 BACKUP-02 BACKUP-03 WSUS-01 SCCM-01 MON-01 AD-SYNC-01 VPN-GW-01 VPN-GW-02 POS-STORE-001 POS-STORE-002 POS-

Three jobs. On every box.

Install	Configure	Scripts



Job 1: Install

Next → Next → Finish

...times two hundred. **No.**

So: **silent MSI.**

```
msiexec /qn /i NSCP-0.14.1-x64.msi ...
```

Every flag is now something you have to get right ***before*** you touch 200 machines.

Job 2: Configure

[UNIT: JAX]
[STATUS: ACTIVE]

ini file	Readable. Copy it around. ***Breaks the moment one box is different.***
registry	Works with GPO and provisioning tools. Opaque. ***Nobody reviews a registry diff.***
https	Fetched remotely from a central repo. ***Forshadowing...***
pushed in	Use curl to update config automatically. ***Forshadowing...***

All are **write-once**.

Job 3: Scripts

A script is **two things**:

```
scripts\check_backup.ps1
```

```
[/settings/external scripts/scripts]
check_backup = scripts\check_backup.ps1
```

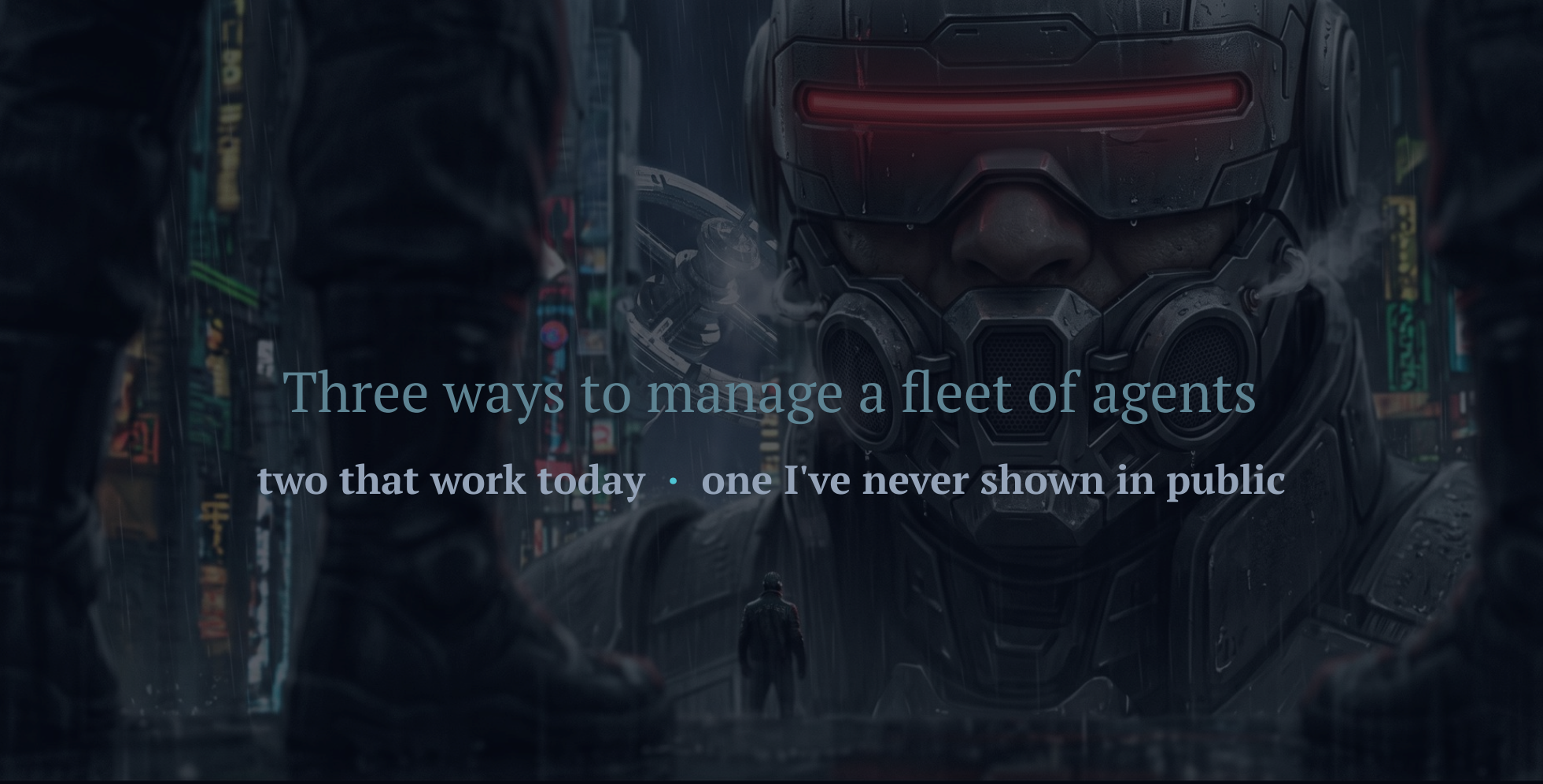
Job 3: Scripts

A script is **two things**:

```
scripts\check_backup.ps1
```

```
[/settings/external scripts/scripts]
check_backup = scripts\check_backup.ps1
```

Which version of `check_backup.ps1` is on which box?

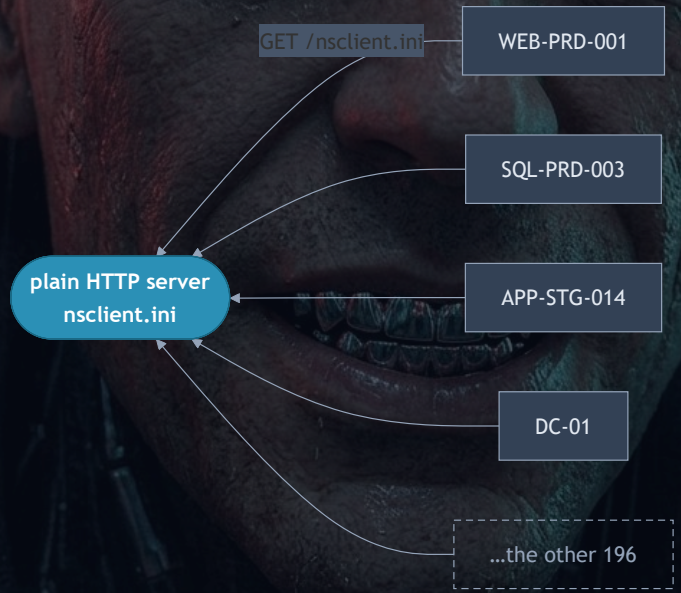


Three ways to manage a fleet of agents

two that work today · one I've never shown in public

1. The config way

The agent drives



One property

```
msiexec /qn /i NSCP-0.14.1-x64.msi ^
    CONFIGURATION_TYPE=https://config.corp.example/nsclient.ini
```

`CONFIGURATION_TYPE` — the settings ***backend***: ini file · registry · **http(s)** · dummy

- refreshed periodically
- changed on the server → picked up by the client
- server unreachable → **keeps using the last config it got**

Not to be confused with `IMPORT_CONFIG` — copies a config ***once***, at install time.

What’s actually on the server

https://config.corp.example/nsclient.ini

```
[/modules]
CheckSystem          = enabled
CheckDisk            = enabled
CheckExternalScripts = enabled
NRPEServer           = enabled

[/settings/default]
allowed hosts = 10.0.0.1

[/settings/NRPE/server]
port = 5666

[/settings/external scripts/scripts]
check_backup = scripts\check_backup.ps1
```

It’s a text file. Put it in git.

And the scripts come too

```
[/attachments]
scripts/check_backup.ps1 = https://config.corp.example/check_backup.ps1
```

The agent downloads it. The registration is in the same file.

Both halves. One place. No drift between them.

Organising the server side

```
config/
  base.ini              ← everything common
  role-web.ini
  role-sql.ini
  site-london.ini
  scripts/
    check_backup.ps1
```

```
[/includes]
base = https://config.corp.example/base.ini
role = https://config.corp.example/role-web.ini
```

First key found wins — **parents override children.**

```
[tls]
verify mode = peer
ca = C:\Program Files\NSClient++\security\ca.pem
```

(boot.ini — since `0.14` `peer` is the default, against the platform CA bundle. Point `ca` at your own CA if the config server isn't publicly signed.)

One config, every host's name

Since `0.17` `${host}` resolves across the whole settings chain — not just the url.

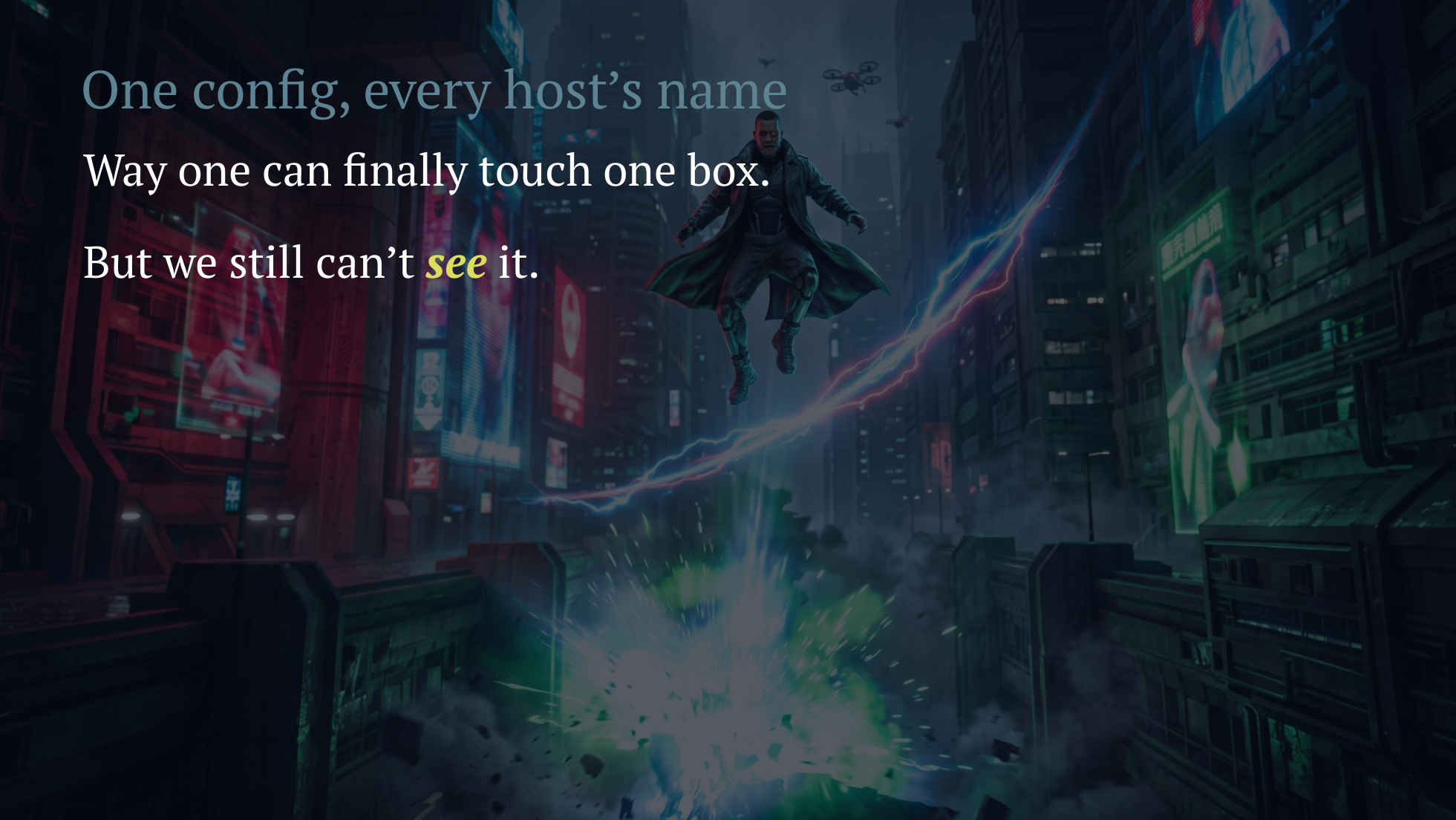
```
[settings]
1 = https://config.corp.example/hosts/${host}.ini

[/includes]
base   = https://config.corp.example/base.ini
client = https://config.corp.example/hosts/${host}.ini

[/attachments]
${shared-path}/${host}.ini = https://config.corp.example/hosts/${host}.ini
```

`${hostname}` · `${host}` · `${domain}` — each with `_lc` / `_uc` variants

- **One boot.ini image for the entire fleet** — every agent asks for ***its own*** file
- A per-host override becomes a file change: update `hosts/web-prd-001.ini` on the server



One config, every host's name

Way one can finally touch one box.

But we still can't ***see*** it.

What you get, what you don’t

You get

- One file controls the fleet. Edit it, everyone follows.
- Machines self-heal on reboot. New machines join by themselves.
- Config is a text file in git.
- No inbound access. Nothing reaches into your boxes.

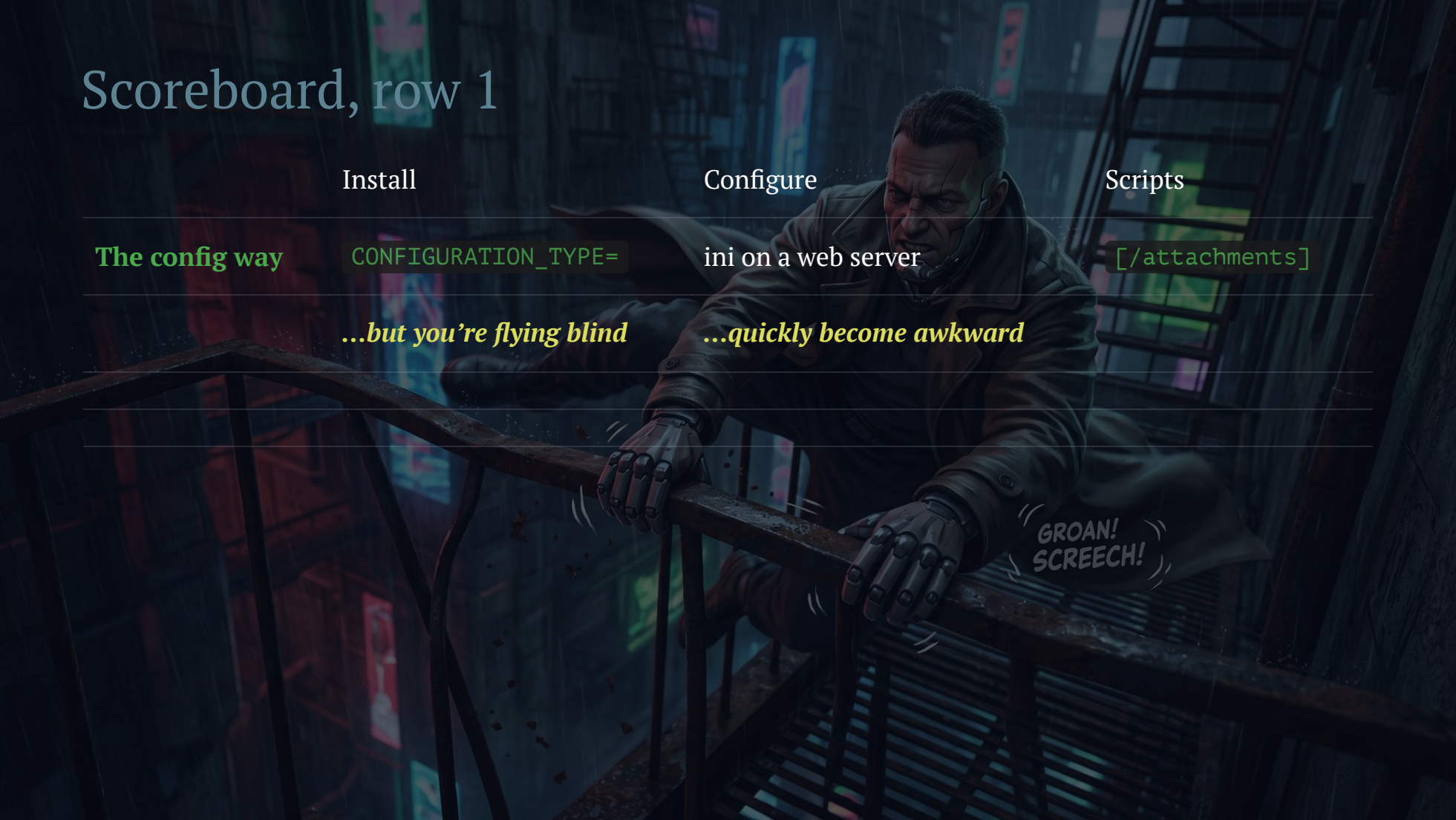
You don’t get

- **No inventory.** Which machines checked in? Which are on the old config?
- **No Security.** There is no real way to manage authentication.
- **Grouping is a convention you invent** and maintain forever.
- **Nothing is pointed at you** when a box goes quiet.

And you can only talk ***at*** a box — never ***with*** it.

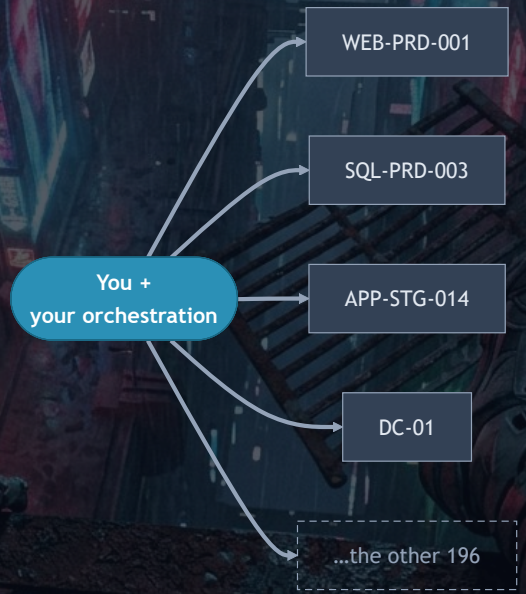
Scoreboard, row 1

	Install	Configure	Scripts
The config way	`CONFIGURATION_TYPE=`	ini on a web server	`[/attachments]`
	...but you're flying blind	***...quickly become awkward***	



2. The hands-on way

You drive



Install it silently

```
msiexec /qn /l* install.log /i NSCP-0.14.1-x64.msi ^
    CONF_WEB=1 ^
    NSCLIENT_PWD=<your API key> ^
    ALLOWED_HOSTS=10.0.0.1
```

`CONF_WEB`	the REST API — ***this is the one that matters today***
`NSCLIENT_PWD`	password / API key
`ALLOWED_HOSTS`	who may connect (empty = **nobody**, since `0.12.3`)

Two accounts, not one

The thing that runs your checks should not be able to rewrite your config.

```
nscp web add-user --user monitoring --password <pw> --role monitoring
```

```
[/settings/WEB/server/users/monitoring]
password = <pw>
role = monitoring
```

Since `0.12.6` — policy, not just credentials:

```
[/settings/permissions]
enabled = true
[/settings/permissions/policies]
WEBServer:admin      = *
WEBServer:monitoring = CheckSystem.check_cpu, CheckSystem.check_drivesize
```

Push the config

```
# set a value
curl -k -u admin -X PUT https://box:8443/api/v2/settings/settings/default \
  -d '{"path":"/settings/default","key":"password","value":"new-secret"}'

# ...writes are queued. persist them:
curl -k -u admin -X POST https://box:8443/api/v2/settings/command \
  -d '{"command":"save"}'
```

```
GET    /api/v2/settings/{path}
PUT    /api/v2/settings/{path}
DELETE /api/v2/settings/{path}?key=     ← separate privilege on purpose
GET    /api/v2/settings/diff?path=/settings/default&recursive=true
```

Push the scripts

```
curl -k -u admin -X PUT \
  https://box:8443/api/v1/scripts/ext/scripts/check_backup.ps1 \
  --data-binary @check_backup.ps1
```

```
Added check_backup as scripts\check_backup.ps1
```

...which writes the registration for you:

```
[/settings/external scripts/scripts]
check_backup = scripts\check_backup.ps1
```

Only scripts under `script root` are visible:

```
[/settings/external scripts]
script root = ${scripts}
```

What it costs you

Every one of these is fixable — in code you write, own, and keep running

- **Inbound access to every Windows box.** Firewall rules. A security team might ask questions.
- **Your orchestration must know every host** — and must be running when the host is.
- **Nothing retries.** Off, rebooting, mid-image — the push fails, and nothing comes back to fix it.
- **No feedback.** Nothing confirms the push landed. Nothing tells you about drift next Tuesday.

None of it is impossible.

It's just a program **you** now have to write.

Scoreboard, row 2

	Install	Configure	Scripts
The config way	`CONFIGURATION_TYPE=`	ini on a web server	`[/attachments]`
	…but you're flying blind	***…quickly become awkward***	
The hands-on way	silent MSI	REST push	REST push
	…and you're the orchestrator		

So I built the missing piece.

NSClient Fleet

A central repository for NSClient config

Not a new monitoring system.

Why this is ***simpler***, not just newer

The inversion

The config way and the hands-on way: **you are the only thing that knows what exists.**

The easy way: **the machines register themselves.**

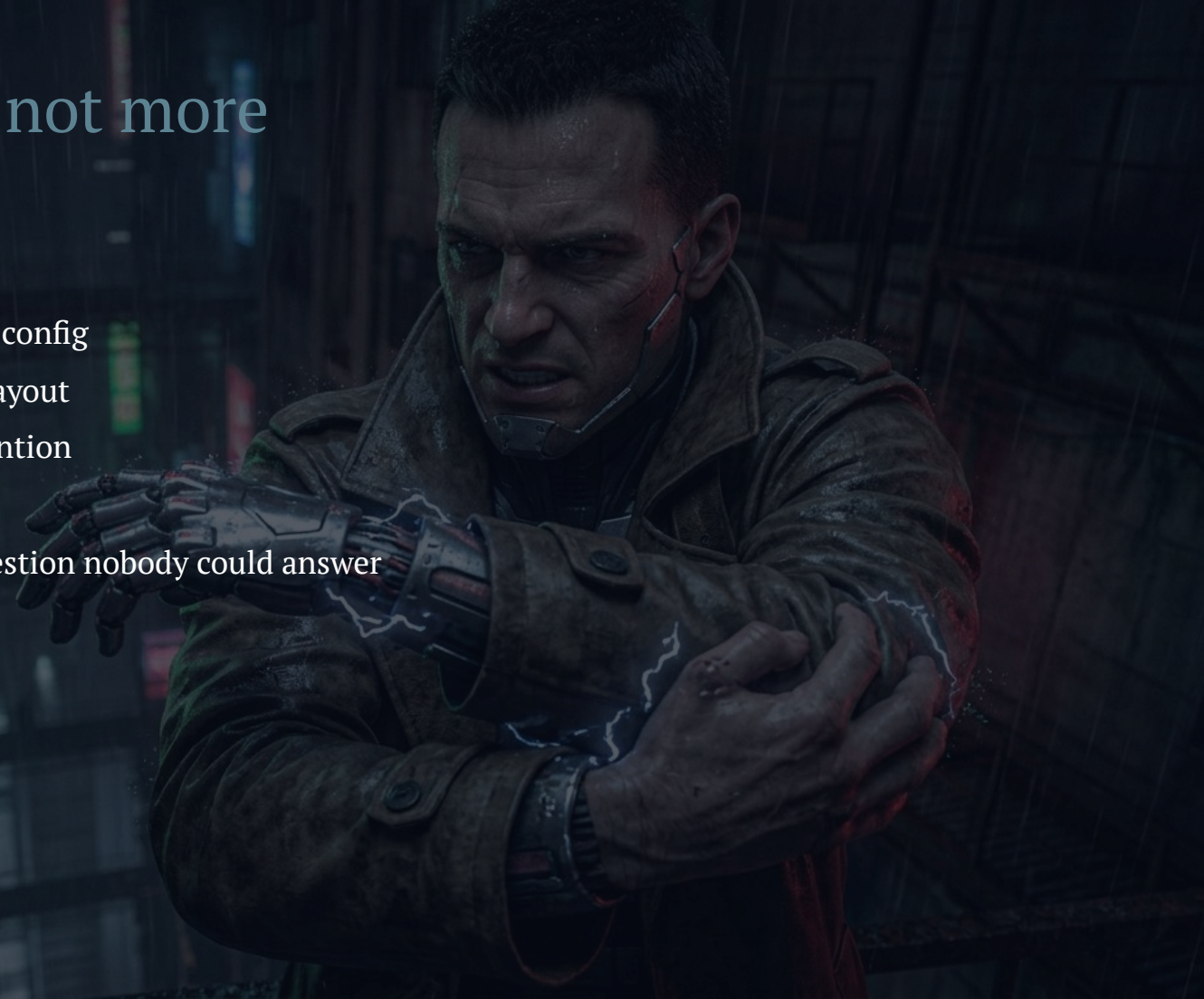
Everything else follows from that one change.

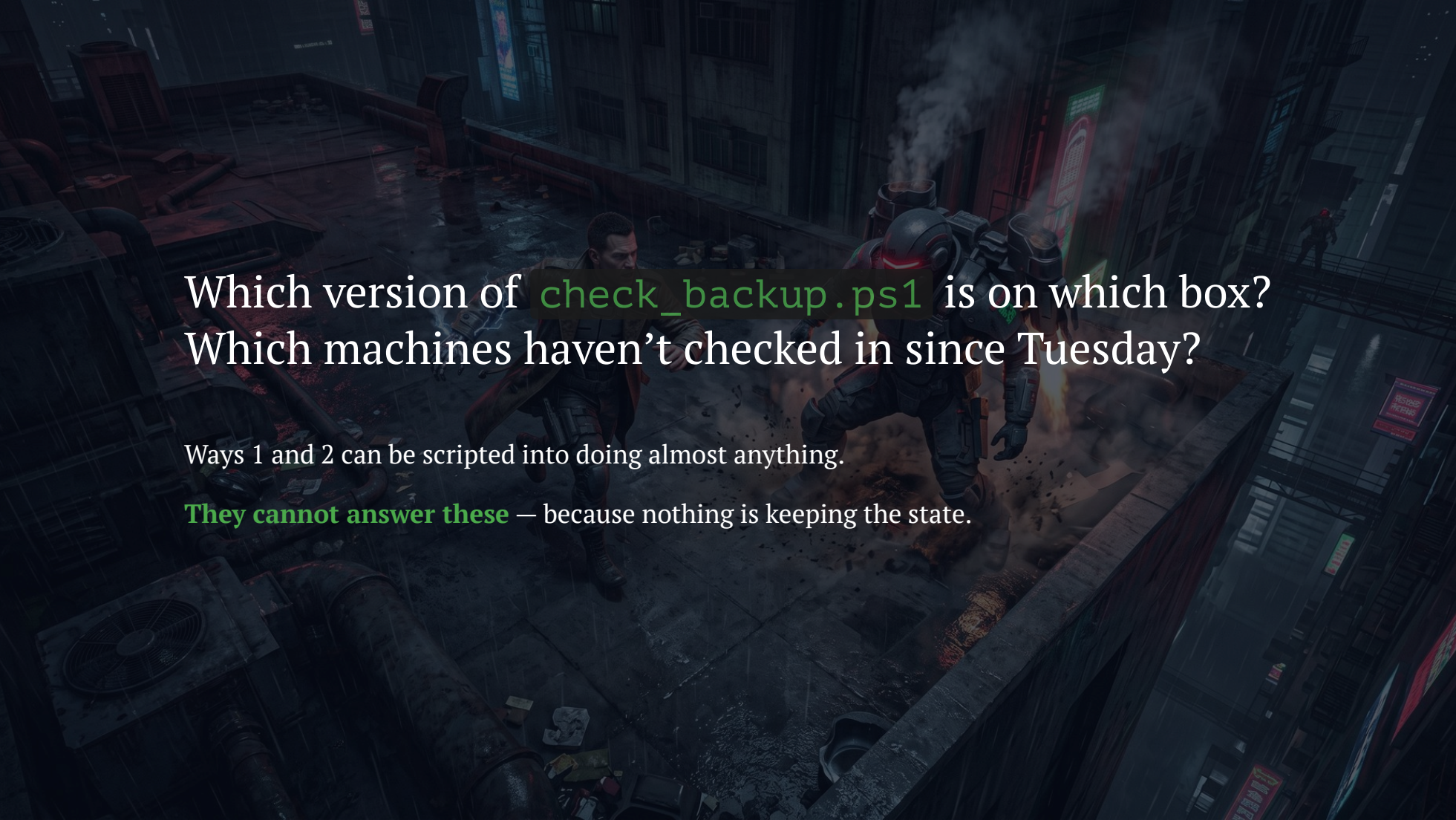
Less machinery, not more

"Great, one more server to run."

It **replaces**:

- the orchestration that pushed config
- the web server and directory layout
- the include-file naming convention
- the inventory spreadsheet
- the "which script is where" question nobody could answer





Which version of `check_backup.ps1` is on which box?
Which machines haven't checked in since Tuesday?

Ways 1 and 2 can be scripted into doing almost anything.

They cannot answer these — because nothing is keeping the state.

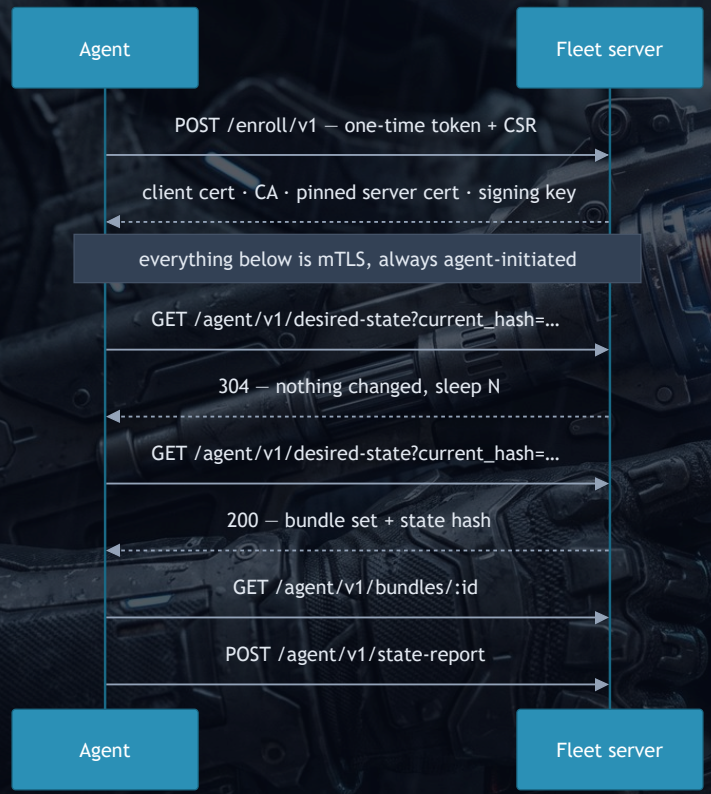
The install (again)

```
msiexec /qn /i NSCP-0.14.1-x64.msi ^
    FLEET_SERVER=https://fleet.corp.example ^
    FLEET_TOKEN=<one-time token>
    FLEET_BUNDLE_KEY=<your key>
```

Same installer. Same shape. Where you had a config URL, you now have a server and a one-time token.

The agent registers itself.

What "registers itself" actually means



Enroll once. After that the agent asks — the server only ever answers.

The boxes describe themselves

The agent reports facts. You don't type them in.

Tag	Where it comes from
`os`	the agent
`os_name`	`Windows Server 2022` — for reading
`os_version`	`10.0.20348` — for matching
`nscp_version`	which agent build is actually on the box
`drives`	`c:,d:` — from CheckDisk
`sqlserver`	`detected` — from the registry, so stopped counts too
your own	map any service to a tag: `MSSQLSERVER=sql-server`

Nobody maintains this list. It maintains itself.

Your own facts, from your own scripts

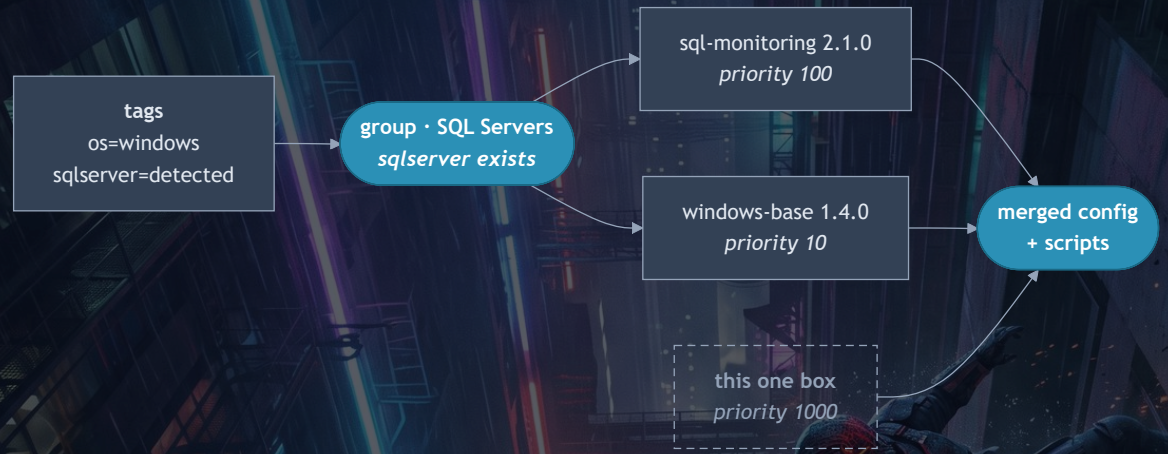
Anything a script can find out, a script can tag.

The script checks	The box now says
is IIS listening on 443?	`role=web-frontend`
which of our apps is deployed here?	`app=billing` · `app_version=4.2`
is there a marker file from provisioning?	`env=lab`
whatever matters to ***your*** fleet	`...=...`

A script's tag works exactly like a built-in one: it feeds the same groups, the same bundles. Stop reporting it, and it goes away on its own.

The built-ins get you started. Your scripts finish the picture.

Tags → groups → bundles



You describe **which machines**, not ***which machine***.

Where bundles come from

New bundle → pick a template → fill in a form

CORRECTION:
THE DRIVE IS PRIMARY!
INTERCEPT AT ALL
COSTS!
(The Leader's voice, grindingly synthetic)

Category	Templates
System health	Windows · Linux · disk space · services & processes · perf counters · real-time alerts
Applications & network	SQL Server · network checks
Monitoring delivery	NRPE · Checkmk · Prometheus · NSCA · NSCA-NG · Icinga 2 · NRDP · Graphite · scheduled baselines
…	…

- One concern per template
- The form edits real INI

The form is a view. The INI is the truth.

…or you make them on your own

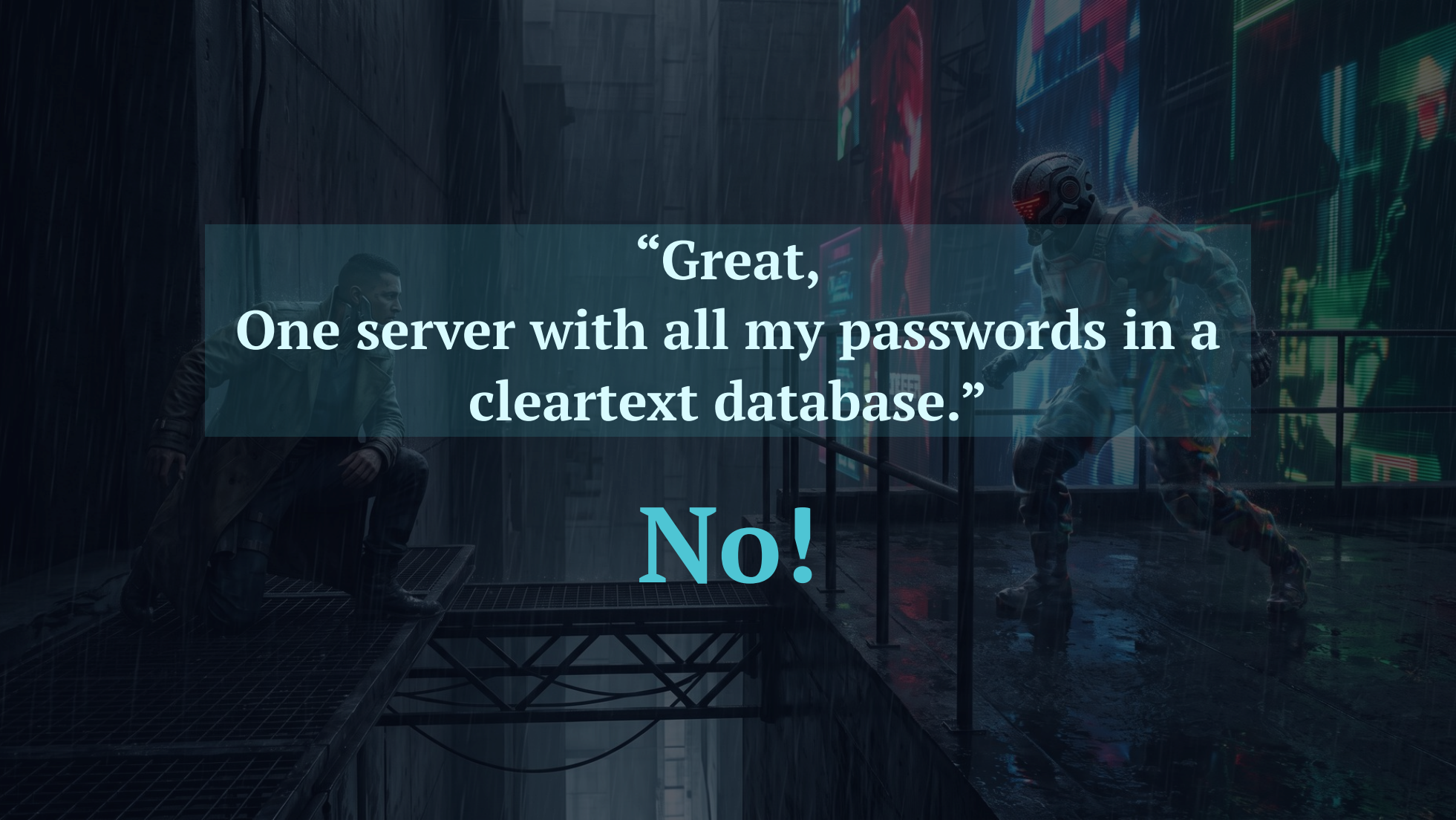
A bundle is a zip. Anything that can make a zip can make a bundle.

```
windows-base-1.4.1.zip
├── bundle.toml        name = "windows-base" · version = "1.4.1"
├── config.json        the config fragment — JSON Merge Patch (RFC 7396)
└── scripts/
    └── check_backup.ps1
```

```
curl https://fleet.corp.example/api/bundles \
  -H "Authorization: Bearer nsk_…" \
  -F name=windows-base -F version=1.4.1 \
  -F bundle=@windows-base-1.4.1.zip
```

Same rules as template-made bundles: objects deep-merge · scalars replace · `null` deletes · script collisions go to the higher priority.

Templates are the on-ramp — the format is yours.


“Great,
One server with all my passwords in a cleartext database.”
No!

Encrypted bundles — sealed before the server ever sees them

Signing protects you from the network. This protects you from the server.

- Encrypted **in your browser** before upload — AES-256-GCM, with a key the server never holds
- The server stores an 8-byte key fingerprint. That's everything it knows.
- It still signs, stores, and delivers the blob — it just **can't open it**.
- The bundle's **name and version are baked into the encryption** — a compromised server can't hand your agents bundle A relabelled as bundle B
- The key reaches agents at install time, out-of-band — same channel as the enrollment token

And the switch: `require encrypted bundles` — the agent refuses anything unsealed. Now bundle content, ***scripts included***, can only come from a key holder. The server can't push code to your boxes even if it wanted to.

Encrypted bundles — the trade

The trade: ***lose the key***, ***lose those bundles***.

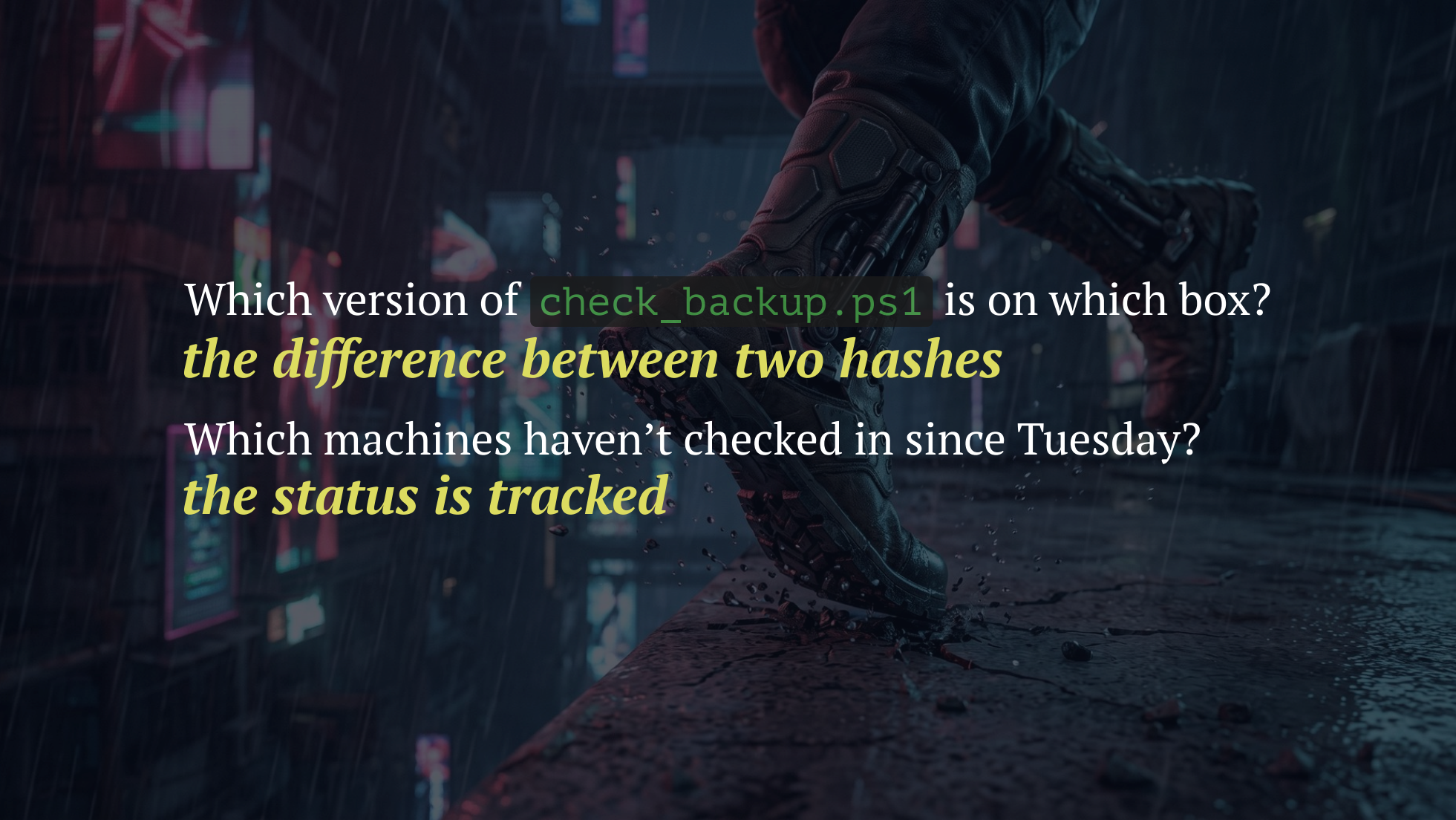
No escrow!

Status

Status	Meaning	Where it comes from
in sync	running exactly what the server would send it	the hash the agent reported = the hash the server would serve **now**
out of sync	change pending — or an apply **failed**	hashes differ; an agent never reports a state it didn't fully apply
offline	quiet for three poll intervals	a reboot or a blip; may fix itself
lost	quiet for 48 hours	stopped, uninstalled, firewalled, or the machine is gone

And one honest flag: **local config** — the host reported it carries settings of its own that ***outrank*** what we send. In sync, and still not necessarily running what you see.

Drift isn't detected. It's the difference between two hashes.



Which version of `check_backup.ps1` is on which box?

the difference between two hashes

Which machines haven't checked in since Tuesday?

the status is tracked

One port, two certificates

443 is the only port. The operator UI, the agents, and certificate issuance all share it.

The client offers	It gets
ALPN `nsclient-fleet/1`	pinned self-signed cert · **client certificate required**
anything else	Let's Encrypt cert · no client certificate requested
ALPN `acme-tls/1`	throwaway challenge cert — handshake only

- Agents pin **one** certificate. They do not use the system trust store.
- So agent connectivity never depends on Let's Encrypt — which is why on-prem and air-gapped behave identically.
- Browsers are never asked for a client cert: no picker dialog, no tenant names leaked to visitors.

The trade: anything that terminates TLS in the middle breaks it.

Nothing is trusted because it asked nicely

Identity is the certificate

- the tenant comes from the certificate's **issuer** — never from what the certificate claims about itself
- the host id is a `spiffe://` name in the certificate, cross-checked against that issuer
- the server builds the certificate itself; your request contributes a public key and nothing else
- no host id, no token, no identity in any request body — ever

Config and scripts are signed

- SHA-256 for integrity, Ed25519 over the digest for authenticity
- the signing key is separate from the CA — compromising one does not forge the other
- every download re-checks membership: a compromised host cannot pull another group's scripts

An agent that can't verify a bundle keeps running the old one.

Demo

Bundles — the checks, as a form

Bundles

A bundle is a zip (manifest + config patch + scripts) signed by your tenant key. Versions are immutable — to roll back, assign the older version to the gro
contents or the key.

Edit windows-server-health@1.0.0 → new version

Name: windows-server-health

Version: 1.0.1

☐ Encrypted

template: Windows server health

VISUAL | INI

CPU check

command: check_cpu time=5m "warn=load > 80" "crit=load > 90"

warn/crit on average load over the time window

Memory check

command: check_memory "warn=free < 20%" "crit=free < 10%"

Disk check

command: check_drivesize drive=* "filter=type in ('fixed')" "warn=free < 20%" "crit=free < 10%"

checks every fixed drive

Uptime check

command: check_uptime "warn=uptime < 2d"

alerts on recently rebooted machines

Save as new version | Cancel

Saved bundles are immutable — saving creates a new (name, version) that you can then assign to groups. Values are written to the agent's `fleet.ini` exactly as typed.

Bundles — services become tags

Bundles

A bundle is a zip (manifest + config patch + scripts) signed by your tenant key. Versions are immutable — to roll back, assign the older version to the grou
contents or the key.

Edit detect-services@1.0.0 → new version

Name: detect-services

Version: 1.0.1

☐ Encrypted

template: Detect installed services

VISUAL | INI

Windows services

Each row maps a Windows service (its short name, as in services.msc) to the tag to publish. While the service is running the host carries <tag> = enabled; when it is stopped or absent the tag is removed. Give every service its own tag. SQL Server is also detected from the registry on every Windows host as sqlserver = detected, running or not.

Service	Tag
DNS	dns-server
SQLSERVERAGENT	sql-server-agent
Spooler	print-server

Add service

Linux systemd units

Same idea for Linux: unit name (without .service) to tag. Unit names vary by distribution — apache2 on Debian/Ubuntu is httpd on RHEL — so add the one your hosts actually use.

Unit	Tag
No units yet — add one below.	

Add unit

Save as new version Cancel

Saved bundles are immutable — saving creates a new (name, version) that you can then assign to groups. Values are written to the agent's `fleet.ini` exactly as typed.

Bundles — or just write the INI

Bundles

A bundle is a zip (manifest + config patch + scripts) signed by your tenant key. Versions are immutable — to roll back, assign the older version to the group. E
contents or the key.

Edit foo@1.0.0 → new version

Name: foo

Version: 1.0.1

☐ Encrypted

template: Windows server health

VISUAL | INI

```
[/settings/check helpers/alias]
foo=check_cpu time=5m "warn=load > 80" "crit=load > 90"
```

Save as new version Cancel

Saved bundles are immutable — saving creates a new (name, version) that you can then assign to groups. Values are written to the agent's `fleet.ini` exactly as typed.

Bundles — secrets go in encrypted

Bundles

A bundle is a zip (manifest + config patch + scripts) signed by your tenant key. Versions are immutable — to roll back, assign the older version to the gr
contents or the key.

Edit passwords@1.0.0 → new version

Name: passwords

Version: 1.0.1

Encrypted

template: Windows server health

VISUAL | INI

```
[/settings/default]
password=secret
```

Encrypt & save as new version

Cancel

Saved bundles are immutable — saving creates a new (name, version) that you can then assign to groups. Values are written to the agent's `fleet.ini` exactly as typed.

Groups — a rule over tags

Groups — and the host decides it belongs

Groups

A group is a saved rule over host tags. Bundles assigned to a group apply to every host the rule matches.

foo

```
print-server = "enabled" [any]
```

⚠ Hosts can join this group by reporting the tag themselves, and will then be served its bundles.

Assigned bundles

foo@1.0.0 · p100 ⊗

— pick a bundle —

Priority 100

Assign

Hosts — enrolling a new machine

Hosts

Refresh + Add host

Enroll the new host

The token expires in 1 hour and can be used once. Run either command on the host, or paste the token into whatever provisions it. Both commands include this tenant's bundle encryption key.

Bootstrap token

For the agent's enroll command, the MSI property, or your own tooling.


```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJob3N0X2lkIjoiMDFNMk1CNlk4WkoxMTY0NVdGVFBFUE5NQTIiLCJ0ZW5hbnRfaWQiOjEsIm5vbmNlIjoiS2E5NW90TDJjYnFUbUtzcV9YUGFWZkNfM3JlVUliaTBSSXBoZllqUy1IayIsImV4cCI6MTc4OTU0
```


Installed agent (command line)

On a host that already has NSClient++ — Windows or Linux.


```
nscp enroll --server https://localhost:9443 --token eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJob3N0X2lkIjoiMDFNMk1CNlk4WkoxMTY0NVdGVFBFUE5NQTIiLCJ0ZW5hbnRfaWQiOjEsIm5vbmNlIjoiS2E5NW90TDJjYnFUbUtzcV9Y
```


Windows installer (MSI)

Installs and enrolls in one step; replace the MSI file name. For a self-signed server certificate add `FLEET_CA=C:\path\to\fleet-ca.pem`, or on a trusted network `FLEET_VERIFY_MODE=none FLEET_INSECURE=1`.


```
msiexec /qn /i NSCP-<version>-x64.msi FLEET_SERVER="https://localhost:9443" FLEET_TOKEN="eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJob3N0X2lkIjoiMDFNMk1CNlk4WkoxMTY0NVdGVFBFUE5NQTIiLCJ0ZW5hbnRfaWQiOjE
```


host_id: 01M2MB6Y8ZJ11645WFTPEPNMA2

Status

The agent enrolls

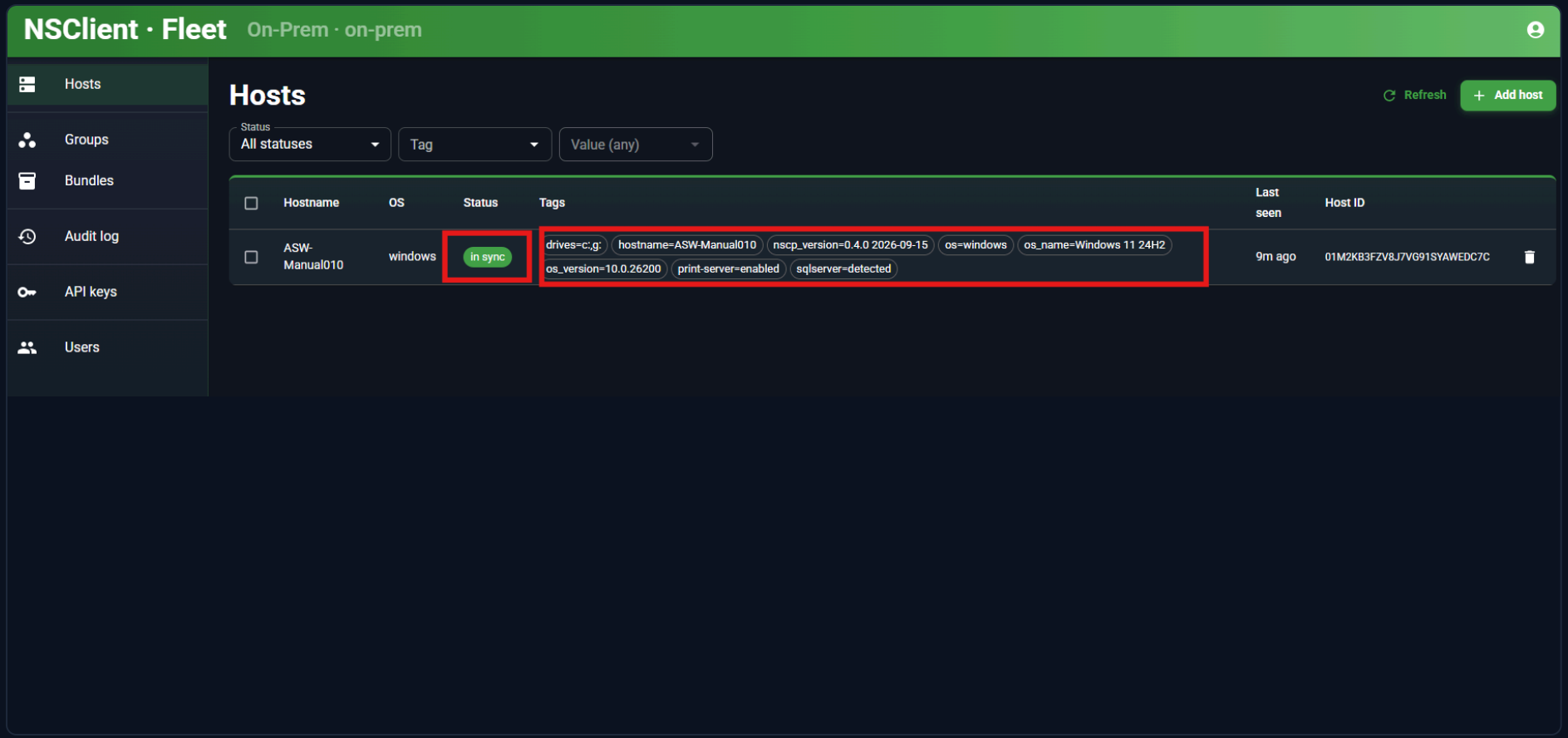
```
: nscp enroll --server https://localhost:9443 --token eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJob3N0X2lk
IjoiMDFNMk1CQkFGVkpONFBSQVhKMFo1WENGWE0iLCJ0ZW5hbnRfaWQiOjEsIm5vbmNlIjoiVG5Ca2pudkpOeWRlRzByeHJfUTlGX1R
pRV9Uel96VVp5NDZoa3VtdWdkRSIsImV4cCI6MTc4OTU0MDQ3MiwiaWF0IjoxNzg5NTM2ODcyLCJhdWQiOiJuc2NsaWVudC1mbGVldC
9lbnJvbGwifQ.uE4B1HuxWhtqsm_aUGwMpkL32PPdaAkd5KgEeYa7mBo --bundle-key sZEslH7/8iQzSd+aYyX0ALrftGQ0nIhxZ
tUVaBQMW/8=
Enrolling with https://localhost:9443...
Enrollment successful.
  Identity stored in: C:\src\nscp\cmake-build-relwithdebinfo-visual-studio/security/agent-state.json
  Agent API (mTLS):   https://localhost:9443
  Bundle keys:        1 (fingerprint 7805e7903e193599)
  Encrypted bundles:  not required
  Fleet configuration sync starts on the next service start.

:
```

First start — the config lands

```
: nscp test
D       core NSClient++ 0.4.0 2026-09-15 x64 Loading settings and logger...
D       core Exported 63 Windows ROOT certificates to C:\src\nscp\cmake-build-relwithdebinfo-visual-studio/security/
windows-ca.pem
D   settings No entries found looking in (adding default): C:\src\nscp\cmake-build-relwithdebinfo-visual-studio/boot
.ini
D   settings Activating: ini://${shared-path}//nsclient.ini
D   settings Creating instance for: ini://${shared-path}//nsclient.ini
D   settings Loading: C:\src\nscp\cmake-build-relwithdebinfo-visual-studio//nsclient.ini
D   settings Creating instance for: ://${fleet-folder}/fleet.ini
D   settings Loading: C:\src\nscp\cmake-build-relwithdebinfo-visual-studio/fleet/fleet.ini
D       core NSClient++ 0.4.0 2026-09-15 x64 booting...
D       core Booted settings subsystem...
D       core Archiving crash dumps in: C:\src\nscp\cmake-build-relwithdebinfo-visual-studio/crash-dumps
D       core permissions: loaded 0 rule(s), enabled=false
L      fleet Fleet configuration sync started (manifest: C:\src\nscp\cmake-build-relwithdebinfo-visual-studio/securi
ty/agent-state.json)
D      fleet Bundle encryption keys configured: 7805e7903e193599
D       core NSClient++ - 0.4.0 2026-09-15 Started!
D       core Loading module C:\src\nscp\cmake-build-relwithdebinfo-visual-studio\modules\CommandClient.dll ()
D       core Loading plugin: CommandClient...
L cli        Enter a command, tab to complete, up for history, help for help, exit to leave.
D fleet      New desired state 3a7cb8abf6212f20c901aa9fd7846fc289a9f2cb7a0a17cf487562b23ca2a383 with 3 bundle(s)
L fleet      Applied fleet configuration 3a7cb8abf6212f20c901aa9fd7846fc289a9f2cb7a0a17cf487562b23ca2a383
D core       Reloading all modules.
D settings   Loading: C:\src\nscp\cmake-build-relwithdebinfo-visual-studio/fleet/fleet.ini
D core       Loading plugin: CommandClient
D core       permissions: loaded 0 rule(s), enabled=false
D core       Found: CheckDisk
D core       Found: CheckHelpers
D core       Found: CheckSystem
```

The fleet view — in sync, and tagged



On the box — what it’s actually running

```
D w32system  Loading counter: disk_queue_length_0 C: = \\ASW-MANUAL010\PhysicalDisk(0 C:)\% Disk Time
D w32system  Loading counter: disk_queue_length__Total = \\ASW-MANUAL010\PhysicalDisk(_Total)\% Disk Time
D w32system  Loading counter: memory_pages_sec = \Memory\Pages/sec
nscp> settings show
/includes/fleet=${fleet-folder}/fleet.ini
/modules/CheckDisk=enabled
/modules/CheckHelpers=enabled
/modules/CheckSystem=enabled
/settings/check_helpers/alias/alias_cpu=check_cpu time=5m "warn=load > 80" "crit=load > 90"
/settings/check_helpers/alias/alias_disk=check_drivesize drive=* "filter=type in ('fixed')" "warn=free < 20%" "crit=
free < 10%"
/settings/check_helpers/alias/alias_memory=check_memory "warn=free < 20%" "crit=free < 10%"
/settings/check_helpers/alias/alias_uptime=check_uptime "warn=uptime < 2d"
/settings/default/password=***
/settings/system/windows/service-tags/DNS=dns-server
/settings/system/windows/service-tags/SQLSERVERAGENT=sql-server-agent
/settings/system/windows/service-tags/Spooler=print-server
D fleet      New desired state 5dc294914c8ffdad361cc85e7e8da4785f19f518d2034e871cd0b118233bf3eb with 4 bundle(s)
L fleet      Applied fleet configuration 5dc294914c8ffdad361cc85e7e8da4785f19f518d2034e871cd0b118233bf3eb
```

Where it’s rough

What’s experimental

Everything is new and will likely have bugs

What will change under you

Templates will likely change?

Maybe richer language for binding templates to tags?

More features, reports, etc…

Will config written today survive?

Yes, config is owned by NSClient++

What’s currently missing

Upgrades

Ironic: but it is coming…

after modern layout is default

Dependencies

If you have script dependencies…

might never come

Automating service monitoring

One rule. Whole fleet.

```
[/settings/scheduler/schedules/check_service]
command = check_service "filter=start_type = 'auto'" "crit=state != 'started'"
interval = 5m
channel = NRDP/WEB
```

One bundle carrying the check → assigned to a group whose selector is `os = windows` → every Windows box gets it on its next poll.

Including the boxes that don't exist yet.

The scoreboard, complete

	Install	Configure	Scripts
The config way	`CONFIGURATION_TYPE=`	ini on a web server	`[/attachments]`
	...but you're flying blind	***...quickly become awkward***	
The hands-on way	silent MSI	REST push	REST push
	...and you're the orchestrator		
The easy way	`FLEET_SERVER=...`	**it's managed**	**it's managed**

Try it without betting anything

I'm not asking you to migrate.

- **Nothing you have breaks.** Everything you do today keeps working.
- **Five machines, not five hundred.** Dev. Lab. The boxes you don't get paged for.
- **They coexist.** Nsclient.ini is not touched (will be reported as local config)
- **The exit is a one-liner** Hate it, go back. Nobody rewrites anything.

Run it on something you don't care about…

…and tell me where it hurts.

Roadmap

- **Upgrade agents** Will land after the new folder layout.
- **Template vars.** Allow you to configure for instance bounds on a host level.
- **Fleet config management.** Currently, you can rotate bundle key and master ca
- **Encrypt fleet config.** Fleet config (including bundle key) is in **clear text on every machine**.
- **Adapters for Nagios** Would be neat if we could automatically add config to Nagios.
- (and other monitoring systems)

Security

You're about to let something else configure your servers. Fair questions:

API key scope & rotation	No standing key. One-time install token, then a 90-day cert the agent renews itself
TLS between agent & server	mTLS both ways. Agent pins the server cert; server demands a client cert
Can the server run arbitrary code on my box?	**By default yes — that's what script deployment is.** Signed bundles; key ≠ the CA. With `require_encrypted_bundles`: **no** — only a key holder can produce a bundle agents accept
Can the server (or its operator) read my config?	Not if you seal it. Encrypted bundles are opaque to the server — it stores a fingerprint, nothing more
Self-hosted? Air-gapped?	Yes. One binary + SQLite, and agent trust never depends on Let's Encrypt

Thank You!

Questions?

NSClient Fleet

Get it → https://github.com/mickem/nsclient-fleet-server

Tell me what's broken → GitHub issues, discussions or michael@medin.name

https://nsclient.org