Platform documentation

HOX project

Stuart J Mackintosh

Wednesday 03 September 2025

Contents

1 HPS system documentation

1.1 Overview

- **Project introduction** – Purpose of HPS, architecture, intended use cases.
- **Design decisions** – Records of key technical choices and rationale.
- **System components** – Description of major directories and modules.

1.2 Quick start

- **Prerequisites** – Required host operating system, packages, and network setup.
- **Installation** – Deploying `hps-container` with `hps-system` and `hps-config`.
- **First cluster setup** – Initialising a cluster with `cluster-configure.sh`.
- **Booting a host** – PXE boot process and selecting a host profile.
- **Verification** – Checking service status and logs.

1.3 System administration

- **Directory layout** – Locations for configuration, logs, distributions, and packages.
- **Cluster management** – Creating, switching, and editing clusters.
- **Host management** – Adding, removing, and updating host configurations.
- **Distribution and repository management** – Managing ISOs, PXE trees, and package repositories.
- **Service management** – Starting, stopping, and reloading dnsmasq, nginx, and supervisord.
- **Backup and restore** – Protecting and recovering configuration and repository data.

1.4 Functions reference

- Automatically generated from `lib/functions.d/` and other libraries.
- One function per page with purpose, arguments, usage examples, and related functions.

1.5 Advanced configuration

- **Kickstart and preseed templates** – Structure, variables, and customisation.
- **iPXE menus** – How menus are built and extended.
- **Storage provisioning** – SCH node disk detection, reporting, and configuration.
- **Integration points** – Hooks for OpenSVC, monitoring, and external systems.

1.6 Troubleshooting

- **PXE boot issues** – Common causes and fixes.
- **Service failures** – Diagnosing and restarting services.
- **Distribution and repository problems** – Checksums, GPG keys, and synchronisation errors.
- **Network problems** – DHCP conflicts, VLAN configuration, and firewall blocks.

1.7 Development

- **Code layout** – File structure and conventions for scripts and libraries.
- **Adding functions** – Naming, argument handling, logging, and documentation guidelines.
- **Testing** – Using `cli/test.sh` and other test harnesses.
- **Contributing** – Workflow, coding standards, and submission process.

1.8 Appendices

- **Glossary** – Definitions of terms and acronyms used in HPS.
- **Environment variables** – Description of exported variables and their use.
- **Decision records** – Full list of design decisions.
- **Reference configurations** – Example cluster, host, and service configuration files.

1.9 Design decisions

Stub: Summarise key design decisions. Link to full decision records in the reference section.

2 Overview

This chapter introduces the HPS system, its purpose, core architecture, and intended use cases.
It also records the major design decisions that shape the system and defines terminology used throughout.

2.1 Introduction

Stub: Provide a high-level explanation of HPS, its goals, and where it fits into the broader infrastructure platform.

2.2 Terminology

Stub: Define common terms, acronyms, and abbreviations used across the documentation.

2.3 Booting a host

Stub: PXE boot process overview and selecting a host profile.

2.4 First cluster setup

Stub: Use `cluster-configure.sh` to create the first cluster and set its parameters.

3 Quick start

A fast path to installing HPS, configuring a cluster, and booting a node.
This section covers the essentials and assumes no prior HPS experience.

3.1 Installation

Stub: Step-by-step guide to deploy `hps-container` with `hps-system` and `hps-config`.

3.2 Prerequisites

Stub: List hardware, OS, packages, network setup, and permissions needed before starting.

3.3 Verification

Stub: Confirming services are active and hosts are provisioned correctly.

4 Installing HPS

How to install the HPS system on the provisioning node, configure services, and verify readiness.

4.1 Hardware

Depends on the scenario, for example:

- evaluation
- home lab
- small organisation
- production-grade
- maximum performance

4.2 Obtaining HPS

How to acquire HPS source

- HPS Repo link
- Link to ISO downloads

4.3 Dependencies and prerequisites

- Operating system ISO's

4.4 Designing networking and numbering

HPS will manage hosts on a directly connected LAN as it uses lower-level protocols such as AMC address to manage key functions. If HPS is required on another indirectly connected network, then that should have it's own IPN.

When configuring the cluster, make sure to use a network address that doesn;t conflict with anythihg else. HPS should offer ranges that do not conflict with anything that it can detect.

In almost every case, HPS will be implemented on a new network segment.

4.5 Service verification

Stub: Checking that dnsmasq, nginx, supervisord, and other components are running.

4.6 Upgrades and maintenance

Stub: Keeping `hps-system` updated without overwriting configuration files.

4.7 Cluster configuration

Stub: Setting DHCP interface, storage subnets, OS type, and other cluster settings.

4.8 Cluster creation

Stub: Running `cluster-configure.sh` and choosing cluster parameters.

4.9 Distribution management

Stub: Adding ISOs, extracting PXE trees, and maintaining package repositories.

5 Deploying and configuring a cluster with nodes

Creating a cluster, configuring its settings, provisioning nodes, and verifying the environment.

5.1 Host profiles

Stub: Assigning profiles such as SCH, TCH, DRH, and CCH to nodes.

5.2 Node provisioning

Stub: PXE/iPXE boot workflow and automated node configuration.

5.3 Service management

Stub: Controlling dnsmasq, nginx, supervisord, and other HPS services.

5.4 Verification

Stub: Checking that nodes are deployed correctly and services are operational.

5.5 Deployment process

Stub: Steps for installing and integrating the DR node into the cluster.

5.6 Purpose of the DR node

Stub: Explain the role of the DR node in ensuring service continuity.

5.7 Failover and recovery testing

Stub: Procedures for verifying DR readiness and simulating failover scenarios.

6 Deploying the disaster recovery node

Installing and configuring the DR node, synchronising data, and testing recovery procedures.

6.1 Preparation

Stub: Hardware, storage, and network prerequisites for the DR node.

6.2 Synchronisation

Stub: Methods for keeping DR node data in sync with the primary environment.

7 Developer documentation

This section provides support for someone who wants to customise their own HPS or contribute to the HPS development.

7.0.1 Load the HPS host libraries

to source the functions, run this on a node:

bash <(curl -fsSL "http://10.99.1.1/cgi-bin/boot_manager.sh?cmd=bootstrap_initialise_distro")

this calls a remote script which pulls down the functions, that are then sourcable

7.1 Cluster management

Clusters are managed by OpenSVC.

The central config file is generated on the IPS and downloaded on demand by cluster hosts. It is dynamically built based on the cluster config.

7.1.1 OpenSVC

we are using v3

Note:

Docs are incomplete.

- V3 docs: https://book.opensvc.com/a
- V2 docs: https://docs.opensvc.com/latest/

Thing you want to set	Where / How
Agent log path/level	`opensvc.conf` (`log_file`, `log_level`)
Agent TCP listener / Web UI ports	`opensvc.conf` (`listener_port`, `web_ui*`)

Thing you want to set	Where / How
Node local tags for default behavior	`opensvc.conf` (`tags`)
Cluster members / node IPs / names	`cluster.conf`
Service resources (zpool/zvol/fs/ip/share)	`om ... create/set` → lives under `services/*`
Placement rules (tags=storage, nodename=…)	`om mysvc set --kw placement=`… (in service cfg)
Start/stop/provision services	‘om mysvc start stopprovision‘
Distribute service configs to other nodes	`om mysvc push` / `om mysvc sync`

7.1.1.1 Useful commands:

= the defined service name

om print config Prints the config for the service
om config validate Checks the config for the node and reports on errors
om purge purge, unprovision and delete are asynchoronous and do things on all node with a object instance

7.1.1.2 References

7.2 Environment variables

Stub: Explanation of exported variables and their purpose.

7.3 Glossary

7.4 Glossary

Btrfs A modern Linux file system with support for snapshots, pooling, and checksumming. Considered for HPS storage but not selected due to lack of native block device export.

CCH (Compute Cluster Host) A host profile type in HPS representing a node dedicated to compute workloads.

CIDR (Classless Inter-Domain Routing) A method for allocating IP addresses and routing, using a prefix length (e.g. /24) to indicate the network mask.

DHCP (Dynamic Host Configuration Protocol) Network protocol that automatically assigns IP addresses and other network settings to devices on a network.

DHCP interface The network interface on which HPS's DHCP service (dnsmasq) listens to respond to PXE boot and other client requests.

DHCP range The range of IP addresses available for assignment via DHCP in a given network segment.

DHCP reservation A mapping of a specific MAC address to a fixed IP address in the DHCP server configuration.

DHCP server A service that hands out IP addresses and network configuration to clients; in HPS, typically provided by dnsmasq.

DHCP subnet The network segment configuration for DHCP, usually defined by IP address and netmask/CIDR.

DHCP static lease A lease configuration mapping a specific client to a specific IP, without dynamic changes.

DHCP options Additional configuration data sent by DHCP server to clients (e.g., boot file name, domain name, DNS servers).

DHCP vendor class identifier A DHCP field that can identify the client's hardware or software type.

DHCP relay A service that forwards DHCP requests from clients in one network to a DHCP server in another network.

DHCP snooping A network switch feature that limits DHCP responses to trusted ports.

DHCP starvation An attack in which an attacker sends repeated DHCP requests to exhaust the available address pool.

DHCPDISCOVER DHCP message type sent by clients to find available DHCP servers.

DHCPOFFER DHCP message type sent by servers in response to a DHCPDISCOVER, offering an IP configuration.

DHCPREQUEST DHCP message type sent by clients to request offered configuration.

DHCPACK DHCP message type sent by the server to confirm an IP lease to the client.

dnsmasq Lightweight DNS forwarder and DHCP server used by HPS to provide PXE boot and local DNS services.

DR node (Disaster Recovery node) A dedicated node used to provide failover capability and data recovery in the event of a primary node failure.

DRH (Disaster Recovery Host) Host profile type in HPS representing a disaster recovery node.

HPS (Host Provisioning Service) The provisioning framework implemented in this project for automated PXE-based OS deployment and configuration.

HPS container The Docker container providing the HPS services.

HPS config The configuration directory structure for HPS, stored separately from the core scripts to allow upgrades without overwriting site-specific settings.

HPS system The set of Bash scripts, functions, and service configurations that implement the HPS provisioning environment.

iPXE Open-source network boot firmware supporting protocols such as HTTP, iSCSI, and PXE. Used by HPS for dynamic boot menus and provisioning.

ISO (International Organization for Standardization image file) A disk image format commonly used to distribute operating system installation media.

Kickstart Automated installation method used by Red Hat-based distributions, configured via a `.ks` file.

MAC address Media Access Control address, a unique identifier assigned to a network interface.

NFS (Network File System) Protocol for sharing files over a network, not used for boot in HPS but sometimes relevant for Linux provisioning.

PXE (Preboot eXecution Environment) Network boot framework that allows a system to boot from a network interface before an OS is installed.

PXE boot menu The interactive menu shown to PXE/iPXE clients to select a boot option.

SCH (Storage Cluster Host) Host profile type in HPS representing a node dedicated to storage services.

syslog Standardised system logging protocol used to collect logs from services.

TCH (Thin Compute Host) Host profile type in HPS representing a lightweight compute node.

TFTP (Trivial File Transfer Protocol) Simple file transfer protocol used in PXE boot to transfer bootloaders and configuration.

ZFS Advanced file system with volume management, snapshots, and data integrity features. Selected in HPS for iSCSI exports due to native zvol support.

8 Reference

Static technical reference information for HPS, including function documentation, troubleshooting guides, and configuration details.

8.1 Library functions

There are two main libraries of functions, the hps functions, and host functions.

hps functions are used during the cluster build and configure process whereas the host functions are available on the running host.

8.2 Reference configurations

Stub: Example `cluster.conf`, `host.conf`, and service configuration files.

8.3 External resources

Below are external projects, code repositories, and documentation sources that are relevant to HPS.
These provide additional background, tooling, or dependencies that are either directly used or referenced in the design.

8.3.1 Btrfs

Link: https://btrfs.readthedocs.io

Summary: Linux file system with features such as snapshots, compression, and subvolumes. Considered for HPS storage but not selected as the primary iSCSI export filesystem.

8.3.2 dnsmasq

Link: http://www.thekelleys.org.uk/dnsmasq/doc.html

Summary: Lightweight DNS forwarder and DHCP server used in HPS to provide PXE boot services, static leases, and local DNS.

8.3.3 Docker

Link: https://docs.docker.com

Summary: Containerisation platform used to run the HPS environment (`hps-container`) in a controlled, reproducible manner.

8.3.4 iPXE

Link: https://ipxe.org

Summary: Open-source network boot firmware supporting PXE, HTTP, iSCSI, and scripting. HPS uses iPXE binaries (`ipxe.efi`, `undionly.kpxe`, `snponly.efi`) for boot menus and network installs.

8.3.5 ISO images for Rocky Linux

Link: https://download.rockylinux.org/pub/rocky/

Summary: Official Rocky Linux distribution media. HPS uses these ISOs to populate PXE boot trees and perform installations.

8.3.6 Kickstart documentation

Link: https://pykickstart.readthedocs.io

Summary: Red Hat-based distributions' automated installation system. Kickstart files are used in HPS to perform unattended OS deployments.

8.3.7 OpenSVC

Link: https://www.opensvc.com

Summary: Cluster resource manager and service orchestrator considered for integration with HPS for managing ZFS-backed iSCSI storage and service failover.

8.3.8 Pandoc

Link: https://pandoc.org

Summary: Document converter used to compile HPS Markdown documentation into PDF, HTML, and other formats.

8.3.9 PXE specification

Link: https://en.wikipedia.org/wiki/Preboot_Execution_Environment

Summary: Standard network boot process for x86 systems. HPS extends PXE with iPXE for additional protocol support and boot menu scripting.

8.3.10 Rocky Linux

Link: https://rockylinux.org

Summary: Enterprise-grade Linux distribution used as a primary OS target in HPS deployments.

8.3.11 syslog protocol (RFC 5424)

Link: https://datatracker.ietf.org/doc/html/rfc5424

Summary: Standard for message logging in IP networks. HPS services can log via syslog for centralised collection.

8.3.12 TFTP

Link: https://datatracker.ietf.org/doc/html/rfc1350

Summary: Simple file transfer protocol used to serve PXE/iPXE bootloaders and configuration files to network boot clients.

8.3.13 ZFS on Linux (OpenZFS)

Link: https://openzfs.org

Summary: Advanced file system and volume manager selected for HPS storage exports due to native block device (zvol) support, snapshots, and data integrity features.

8.4 Troubleshooting

Stub: Common problems, causes, and fixes for HPS operation.

8.4.1 logging

See the log files / syslog

8.4.2 Function test

env QUERY_STRING="cmd=generate_opensvc_conf" bash -x /srv/hps-system/http/cgi-bin/boot_manager.sh

8.4.3 HPS system documentation

8.4.3.1 Overview

- **Project introduction** – Purpose of HPS, architecture, intended use cases.

- **Design decisions** – Records of key technical choices and rationale.
- **System components** – Description of major directories and modules.

8.4.3.2 Quick start

- **Prerequisites** – Required host operating system, packages, and network setup.
- **Installation** – Deploying `hps-container` with `hps-system` and `hps-config`.
- **First cluster setup** – Initialising a cluster with `cluster-configure.sh`.
- **Booting a host** – PXE boot process and selecting a host profile.
- **Verification** – Checking service status and logs.

8.4.3.3 System administration

- **Directory layout** – Locations for configuration, logs, distributions, and packages.
- **Cluster management** – Creating, switching, and editing clusters.
- **Host management** – Adding, removing, and updating host configurations.
- **Distribution and repository management** – Managing ISOs, PXE trees, and package repositories.
- **Service management** – Starting, stopping, and reloading dnsmasq, nginx, and supervisord.
- **Backup and restore** – Protecting and recovering configuration and repository data.

8.4.3.4 Functions reference

- Automatically generated from `lib/functions.d/` and other libraries.
- One function per page with purpose, arguments, usage examples, and related functions.

8.4.3.5 Advanced configuration

- **Kickstart and preseed templates** – Structure, variables, and customisation.
- **iPXE menus** – How menus are built and extended.
- **Storage provisioning** – SCH node disk detection, reporting, and configuration.
- **Integration points** – Hooks for OpenSVC, monitoring, and external systems.

8.4.3.6 Troubleshooting

- **PXE boot issues** – Common causes and fixes.
- **Service failures** – Diagnosing and restarting services.
- **Distribution and repository problems** – Checksums, GPG keys, and synchronisation errors.
- **Network problems** – DHCP conflicts, VLAN configuration, and firewall blocks.

8.4.3.7 Development

- **Code layout** – File structure and conventions for scripts and libraries.
- **Adding functions** – Naming, argument handling, logging, and documentation guidelines.
- **Testing** – Using `cli/test.sh` and other test harnesses.
- **Contributing** – Workflow, coding standards, and submission process.

8.4.3.8 Appendices

- **Glossary** – Definitions of terms and acronyms used in HPS.
- **Environment variables** – Description of exported variables and their use.
- **Decision records** – Full list of design decisions.
- **Reference configurations** – Example cluster, host, and service configuration files.

8.4.4 `bootstrap_initialise_distro`

Contained in `lib/functions.d/configure-remote-host.sh`

Function signature: 1664d8a7eb2277600c48a3bc6974c34ea586df3fb5c9bfb5b7b8268285d91c63

8.4.5 Function overview

This function, `bootstrap_initialise_distro()`, is utilised for initializing the process of bootstrapping on a specific distribution. It primarily utilizes a local variable `mac` which is passed as an argument. The function then runs a bash script, the output of which is printed out using the `cat` command.

8.4.6 Technical description

- **name**: `bootstrap_initialise_distro()`
- **description**: This function's main use is to initialize the bootstrapping process in a specific Linux distribution. The function takes MAC address as input which specifies the target machine for bootstrapping. The body of the function is a bash script that is implying an offline bootstrapping process from a provisioning server.
- **globals**: None
- **arguments**:
  - `$1: mac` - This argument represents the MAC address of the target machine.
- **outputs**: The output of the function is a printed bash script meant to conduct an offline bootstrap process from a provisioning server.
- **returns**: The function does not have a explicit return value since it's mainly a bash script output. The function's job is to print out a bash script.
- **example usage**:

```
bootstrap_initialise_distro "00:0a:95:9d:68:16"
```

8.4.7 Quality and security recommendations

1. Security can be improved by validating the MAC address input to ensure it's formatted correctly and exists in the network.
2. Validate the user's privilege level before running the script to safeguard against unauthorized access.
3. Better error handling could be implemented to account for any issues during the script's operation.
4. Avoid storing sensitive information such as passwords in the script to enhance security.
5. Include a logging functionality that keeps a record of changes made by the script for auditing and debugging purposes.

8.4.8 `build_yum_repo`

Contained in `lib/functions.d/repo-functions.sh`

Function signature: 55fba2752d840b45670dcc10f8be084d3234f9c87a37a1959fcde6a9247be828

8.4.9 Function Overview

The `build_yum_repo` function checks for changes in RPM packages inside a specified repository. If changes are detected or if there's no previous state, it uses the `createrepo_c` command to create or update the metadata for the YUM repository. Regardless of the outcome, a checksum of current state is stored for future comparison. Outputs and potential errors are reported using `hps_log` helper function.

8.4.10 Technical Description

- **Name:** `build_yum_repo`
- **Description:** The function checks the RPM changes inside the provided repo path. If there are changes or no previous state, `createrepo_c` is used to create or update the repo metadata. Checksums of the current state are saved for future checks.
- **Globals:** None
- **Arguments:** `repo_path` ($1): the path to the repo that needs to be checked and updated.
- **Outputs:** Associated logs with `hps_log` displaying info about events and potential errors.
- **Returns:** 0 if no changes were made or repo created successfully, 1 if the repo path is not provided or doesn't exist, 2 if `createrepo_c` command is not found.
- **Example usage:** `build_yum_repo "${HPS_PACKAGES_DIR}/${DIST_STRING}/Repo"`

8.4.11 Quality and Security Recommendations

1. Consider validating the checksum process and handling any exceptions it might throw. This can improve the function's quality and resilience.
2. Assert the existence of 'createrepo_c' command at the start of the function to fail early if it's not installed, this can save execution time.
3. Look into the security of the checksum generation. Make sure it is robust against potential risks such as spoofing or collision attacks.
4. Validate the structure and content of the repo path argument to prevent potential bugs or security issues related to wrong or malicious inputs.

8.4.12 `build_zfs_source`

Contained in `lib/host-scripts.d/rocky.sh`

Function signature: c64e26c5e886b1a0aded061414c66873630e97cae41cf2c7a37f800fd6866674

8.4.13 Function Overview

This function, `build_zfs_source` is designed to perform several tasks that include building ZFS sources for the Rocky Linux distribution. The steps it follows are quite straightforward. The function retrieves the necessary ZFS source data from a defined URL path, downloads the corresponding file, installs ZFS build dependencies, extracts the source file into a temporary build directory, compiles, and finally, installs it, while logging various parts of the process using the `remote_log` function.

8.4.14 Technical Description

- **Name:** `build_zfs_source`
- **Description:** The function downloads and builds the ZFS source for Rocky.
- **Globals:** None.
- **Arguments:** None.
- **Outputs:** Logs the various steps in the process using the `remote_log` function. Use of this function suggests that the output of this function may be directed to a remote log server or service.
- **Returns:** Returns 0 if the ZFS source is successfully built and installed. Returns 1 if any error occurs at any stage of the process.
- **Example usage:** To use this function, it would be sourced and executed in a Bash shell as follows:

```
. path/to/script.sh
build_zfs_source
```

8.4.15 Quality and Security Recommendations

1. It may be beneficial to add some error handling to ensure that the ZFS source URL is valid and available. This would greatly enhance the stability of the function.
2. The use of `curl` and `wget` to download files could potentially introduce security risks if the source URLs are compromised. Implement SSL/TLS certificate checks to secure downloads.
3. In the `remote_log` calls, sensitive information such as URLs might be logged. Best practices should be put in place to sanitise any sensitive information before logging.
4. The script does something if it can't `curl` the ZFS source file (it tries `wget`), but it doesn't check if it can resolve the host or even access the network in the first place. An initial network check could be implemented to enhance the resiliency of the Bash function.
5. The script doesn't verify the downloaded file's integrity. One way to do this is to compare the checksum of the downloaded file to a known good one.
6. The function uses a known temporary workspace i.e., '/tmp/zfs-build'. It should instead create a dynamic secure temporary workspace to prevent potential file conflicts and security issues.

8.4.16 `cgi_auto_fail`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: dc511ce0952b232a2422ca3ae6f4600d46e70a01062014017e66599f7fc2dc95

8.4.17 Function Overview

The `cgi_auto_fail()` function is used to automatically fail a Common Gateway Interface (CGI) application. The function uses a local message or uses a default message if none is provided. It detects the client type and fails based on that detection. If the client is `ipxe`, it calls the `ipxe_cgi_fail` function. If the client is `cli`, `browser`, or `unknown`, it calls the `cgi_fail` function.

8.4.18 Technical Description

- Name: `cgi_auto_fail`
- Description: A function for automated failure of CGI applications. It detects the client type and fails accordingly, using the `ipxe_cgi_fail` for `ipxe` clients and `cgi_fail` for `cli`, `browser`, or `unknown` clients.
- Globals: None
- Arguments:

  - `$1`: An optional argument. This represents a custom fail message to be used. If no message is provided, a default error message is used.
- Outputs: The function outputs an error message to the shell.
- Returns: Nothing.
- Example Usage: `cgi_auto_fail "Failed to load the webpage.` This will output an error, “Failed to load the webpage,” and detect the client type to fail accordingly.

8.4.19 Quality and Security Recommendations

1. Ensure that input validation is done before calling the function to avoid unexpected results or error messages.
2. Maintain the mappings for failure functions for each client type in one place for easy updates and adjustments.
3. Make sure the function handles all possible client types to ensure robustness.
4. Improve the default error message to provide more context or guidance to the user.
5. When handling errors in web applications, it’s critical to manage the information that you expose to the user. Therefore, ensure that the error information does not expose any sensitive information like system details or user data.
6. Make sure you always update your systems and monitor for any security vulnerabilities or attacks.

8.4.20 `cgi_fail`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: 5bc8c57b97b640ef4a194c0065da11fec44b1f1bb0525bf46e8163d2a50cabd5

8.4.21 Function Overview

The function `cgi_fail` depicted above performs the essential role of logging an error message and outputting it through the server’s CGI header. It takes the error message as an argument and performs two main actions: logging the error message using `hps_log` function and outputting the error message. This function can come handy in cases where you want to keep track of all the errors that occur on your server, and also display them on the server for debugging purposes.

8.4.22 Technical Description

In a more detailed technical perspective, the function’s definition is structured as follows:

- **Name:** `cgi_fail`

- **Description:** This function logs an error message to the system and outputs it through the server's CGI header. It operates by invoking the `cgi_header_plain` function and the `hps_log` function.
- **Globals:** None
- **Arguments:**
  - `$1:` An error message to log and output through the CGI header.
- **Outputs:** This function debugs the provided message and prints it to the standard output.
- **Returns:** Does not return any explicit value.
- **Example Usage:** To use `cgi_fail`, you would generally pass it an error message as such: `cgi_fail "Unknown error encountered"`.

8.4.23 Quality and Security Recommendations

1. **Input Validation:** Given that the function is accepting user-provided strings, the function should perform proper input validation on the error message to prevent possible injection attacks.
2. **Error Handling:** The function does not contain any explicit error handling. Implementing checkpoints to verify successful suppression of errors might be useful.
3. **Logging:** Consider adding timestamps and additional context to the logged errors for ease of troubleshooting.
4. **Fallbacks:** It may be beneficial to add a fallback or default error message if a blank error message is provided.
5. **Code Documentation:** Make sure to document what the function does and how to use it. Include details of what each argument should be and what the function returns. This can help others to use the function correctly and avoid mistakes.

8.4.24 `cgi_header_plain`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: cce7eb1544681966ec11d5f298135158497fc2ac56c89b00c63c84b8e1bc733a

8.4.25 Function Overview

The `cgi_header_plain` function is a Bash shell function that is designed to produce the HTTP header for a plain text response. In a CGI (Common Gateway Interface) context, this function can be used to clearly define the content type of the output as plain text.

8.4.26 Technical Description

```
def cgi_header_plain:
    - name : cgi_header_plain
    - description : A Bash function that prints the HTTP header for
↪   a plain text response, typically used in a CGI context.
```

```
    - globals : None
    - arguments : None
    - outputs :
        - "Content-Type: text/plain"
        - A line break
    - returns : None
    - example usage :
        ```
        #!/bin/bash
        cgi_header_plain
        ```
```

8.4.27 Quality and Security Recommendations

1. Consider checking the status of the `echo` commands to ensure that they successfully sent the outputs.
2. If this function is used in a larger script, make sure that it's being called appropriately and that the output is being properly used.
3. Ensure secure execution with proper user privileges. Running the script with unnecessary admin rights can be a security risk.
4. Audit and keep a record of function usage in order to trace and troubleshoot any issues when they occur.
5. Include error handling to respond to failed operations and cleanly exit the script when necessary. This will prevent any misuse or security vulnerabilities.

8.4.28 `cgi_log`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: 9f2c2cf7c0d57e85a08611717b5d691eddf235f096bbc311bf9d58541f0c77b3

8.4.29 Function overview

The function `cgi_log()` is designed to output log messages with a timestamp into a file. It takes a string as an argument and appends it to the log file with a timestamp for traceability purposes.

8.4.30 Technical description

- **name:** cgi_log
- **description:** The function accepts a string as an input and writes it to the log file (/var/log/ipxe/cgi.log) with a timestamp. This provides a chronological record of all the logging information.
- **globals:** None
- **arguments:**

    - [$1: msg] Log message as string
- **outputs:** Appends the log message with a timestamp to the /var/log/ipxe/cgi.log.
- **returns:** Not applicable.
- **example usage:** `cgi_log "This is a test message"`

8.4.31 Quality and security recommendations

1. **Input Validation**: Always validate the input (msg) before using it. This will prevent log injection attacks.
2. **Log Rotation**: To manage the size of the logs properly, implement some type of log rotation either via a Bash script or a system utility.
3. **Permissions**: Ensure appropriate permissions are set on the log file to prevent unauthorised access or alteration of the logs.
4. **Sensitive Information**: Be cautious while logging messages as it should not include any sensitive data like passwords which can be exposed through logs.
5. **Error Handling**: Consider adding error handling to log any potential errors when trying to write to the log file.

8.4.32 `cgi_param`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: 5007f95c313c04a01df7ba39bff0241f44511cd570d795bc11a890edf032f323

8.4.33 Function overview

The bash function `cgi_param` is designed to decode and retrieve parameters from the `QUERY_STRING` in a CGI context. The function uses a query-string from the CGI context and processes it to detect commands and key-value pairs. The function is triggered to process the query string only once, with subsequent calls to commands (get, exist, equals) retrieving or comparing the saved values. If an unknown command is passed, an error message is returned.

8.4.34 Technical description

- **Name**: `cgi_param`
- **Description**: This function parses a query string into parameters, then provides a way to retrieve a parameter value, check if a parameter exists, or see if a parameter's value matches a provided value by using the 'get', 'exists' or 'equals' commands respectively.
- **Globals**:
    - `QUERY_STRING`: The string to extract parameters and their values from.
    - `__CGI_PARAMS_PARSED`: Global flag to detect if the query string has been parsed.

    - `CGI_PARAMS`: An array to save decoded keys and their values.
- **Arguments**:
    - `$1`: The command to run: 'get', 'exists', 'equals'.
    - `$2`: The name of the parameter.
    - `$3`: Optional. The value to compare against when the command is 'equals'.
- **Outputs**: If the 'get' command is called, the value of the named parameter. If not, a success status or an error message in case of an invalid command.
- **Returns**: The function can return different values depending on the command given. If 'get' command, will print value to stdout. If 'exists', 0 is returned if parameter exists, 1 if not. If 'equals', returns 0 if parameter equals given value, 1 if not. Returns 2 if command is invalid.
- **Example usage**:

```
cgi_param get username
cgi_param exists page_count
cgi_param equals user_role admin
```

8.4.35 Quality and security recommendations

1. Use stricter validation on parameter keys while parsing. Right now, only alphanumeric characters plus underscores are allowed. However, consider more restrictive set.
2. Be sure to escape all variable expansions to prevent code injection.
3. Consider ways of handling or communicating parse failures more explicitly - it may prove difficult to debug if the `QUERY_STRING` is not formatted as expected.
4. Implement a more robust error handling mechanism. For example, when the user provides an invalid command, an exception should be handled that doesn't allow further execution of the script.
5. Add more guard clauses for blank, null, or unexpected inputs to avoid unexpected behaviour.
6. Always try to keep your function's behavior predictable and documented.

8.4.36 `cgi_success`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: 3c468a0d5bf1a1d432f4efcb2a108889f0d0e762f542f50c29b92350f02de3bc

8.4.37 Function Overview

The `cgi_success` function is a utility function in Bash designed for use in CGI scripting. It outputs a plain header using the `cgi_header_plain` function and then echoes out the first argument provided to the function. It acts as a simple mechanism to output a plain text response with a custom message on successful execution of CGI script.

8.4.38 Technical Description

The details of the `cgi_success` function are as follows:

- **Name**: `cgi_success`
- **Description**: This function outputs a plain header and then echoes the first input argument. It is typically used in CGI scripting to output a response with a custom success message.
- **Globals**: None.
- **Arguments**:
  - `$1`: The text to be displayed in the body of the CGI response.
- **Outputs**: Prints out the contents of `$1` following a plain header in CGI response.
- **Returns**: Nothing since `echo` does not have a return value.
- **Example usage**:

```
cgi_success "The CGI Script executed successfully."
```

8.4.39 Quality and Security Recommendations

1. Always validate and sanitize the input passed to the function to avoid a potential injection attack.
2. Implement error checking for the function calls inside `cgi_success` and handle them appropriately.
3. Document the expected values for `$1` clearly for users.
4. To protect against unintended side effects, make sure to quote variables that are referenced to prevent word splitting or pathname expansion.
5. Always use the function in conjunction with proper Header declaration in CGI scripting.

8.4.40 `check_iscsi_export_available`

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: 747cf8b13f4b0b09cc62797344d4a6efa48f7dd4e42c1431fbc4771d9a4058f5

8.4.41 Function overview

The `check_iscsi_export_available` function verifies if all necessary software and hardware components are present and correctly set up in order to export iSCSI targets on a Unix-based system. It does this by validating the presence of the `targetcli` package, checking for a properly mounted `configfs` filesystem, and ensuring the availability of Light-weight Input/Output (LIO) kernel target modules.

8.4.42 Technical description

- **Name**: `check_iscsi_export_available`

- **Description**: This function checks for the presence and configuration of various prerequisites necessary for iSCSI target exporting.
- **Globals**: None
- **Arguments**: No arguments are expected by this function.
- **Outputs**: This function outputs various error or success messages outlining the state of the export environment.
- **Returns**: Returns `0` if all checks pass and the iSCSI export environment is ready (both 'targetcli' and LIO kernel modules are available). Returns `1` and echoes an error message if any of the checks fail.
- **Example usage**:

```
check_iscsi_export_available
```

8.4.43 Quality and security recommendations

1. To improve soundness and maintainability of the function, it is recommended to implement a more robust error handling mechanism. This might include handling for potential issues during the execution of the embedded shell commands.
2. For security purposes, consider controlling permissions (via 'chown', 'chmod', etc.) on `/sys/kernel/config` to ensure only the required processes and users can access and modify it.
3. To further enhance function reliability, consider adding checks for specific versions of the 'targetcli' package and LIO kernel modules, as different versions may not behave identically.

8.4.44 `check_zfs_loaded`

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: e83726e842477cf79c67cdcf1de046b56eacfafd7f97ba195d70b1f2bccfabc2

8.4.45 Function Overview

The `check_zfs_loaded` function checks whether the ZFS (Zettabyte File System) is installed and operational on the machine where the script is running. If the ZFS command isn't found or the ZFS kernel module isn't loaded, the function attempts to perform the necessary actions and gives feedback on the actions performed. The function ultimately returns 0 if the system passes all checks; otherwise it returns 1.

8.4.46 Technical Description

8.4.46.1 Name

```
check_zfs_loaded
```

8.4.46.2 Description

This bash function checks two conditions: if the ZFS command is found on the system, and if the ZFS kernel module is loaded. If any of these conditions is not met, it prompts the user with an error message. Moreover, if the ZFS module is not loaded, it attempts to load it using the `modprobe` function. This operation might require superuser privileges.

8.4.46.3 Globals

None.

8.4.46.4 Arguments

No arguments taken.

8.4.46.5 Output

The function outputs to stdout:

- A successful or unsuccessful message indicating ZFS command installation status.
- A status update on whether the ZFS module is loaded, whether a load attempt was made, and whether that was successful.

8.4.46.6 Returns

- `1`: if either the ZFS command is not found OR the ZFS module failed to load.
- `0`: if the ZFS command is found AND the ZFS module is successfully loaded.

8.4.46.7 Example Usage

```bash
if check_zfs_loaded; then
  echo "ZFS is ready to use."
else
  echo "ZFS is not properly set up."
fi
```

8.4.47 Quality and Security Recommendations

1. Authentication Check: The function should check if the user has required permissions to perform actions such as loading a kernel module using `modprobe`. Otherwise, it could misleadingly signal a failure when the underlying issue is a lack of permission.
2. Error Catching: It might be beneficial to add additional error handling or checking. For example, catching and further diagnosing errors that result from the `command` or `modprobe` functions could provide more clarity as to what caused a failure.

3. Reliable Checking: Checking presence of ZFS only through presence of `zfs` command and loading status of module might not fully confirm its operational status. Integrate a more reliable way of verifying whether ZFS is properly working, such as creating a test file on a ZFS filesystem.
4. Logging: Consider adding logging for better troubleshooting capabilities. Rather than just echoing the status, also write it into a log file.
5. Use Safe Bash Flags: Consider setting safe bash flags (`set -euo pipefail`) at the beginning of your scripts to avoid certain common bash pitfalls.

8.4.48 `configure_kickstart`

Contained in `lib/functions.d/configure_kickstart.sh`

Function signature: b9c974579a4cf0918b1f523ab5546c0fabf4f4fe0d6a0a4ab77a23765ef1cbca

8.4.49 Function Overview

The `configure_kickstart()` function is designed to generate a "Kickstart" file for a given cluster name. The generated Kickstart file is used to automatically install a Linux operating system on a physical or virtual machine. The function checks if a cluster name is provided, then combines several system configurations and packages into a Kickstart configuration file that can be used to quickly deploy a Linux-based cluster.

8.4.50 Technical Description

- **name**: configure_kickstart
- **description**: A bash function that generates a Kickstart configuration file used to automatically install a Linux operating system on a machine (physical or virtual) in a single step without human interaction. It takes the cluster name as an argument.
- **globals**: None.
- **arguments**:
  - *$1*: Cluster name. This argument denotes the name of the cluster which will be installed using the Kickstart file.
- **outputs**:
  - Kickstart installation file located in "/srv/hps-config/kickstarts" directory with name "${CLUSTER_NAME}.ks".
  - Console messages acknowledging the progress of the function.
- **returns**:
  - Exits with code 1 if a cluster name is not provided.
  - Otherwise, no specific return value.
- **example usage**:

```
configure_kickstart "my_cluster"
```

8.4.51 Quality and Security Recommendations

1. Error handling could be improved with more extensive input verification for cluster name before creating files and directories.
2. The script should avoid using hardcoded passwords. A recommended approach might be to use secret management or environment variables to handle authentication.
3. Avoid running applications as the root user. Consider using less privileged users for running applications where possible.
4. There should be a method to update and/or manage packages after the installation. Keeping software/packages updated is an integral part of maintaining the security of the system.
5. It's important to implement logging that can provide insight into the function's operation and make troubleshooting easier. Especially if an issue arises, logs will be crucial in solving it.

8.4.52 `configure_supervisor_core`

Contained in `lib/functions.d/configure-supervisor.sh`

Function signature: 791d70bd40d5321833762eec6e797655e2f05a2aae1ffb9e5e783ccacdf6127a

8.4.53 Function Overview

The function `configure_supervisor_core` creates a configuration file for the supervisor program at the location defined by `SUPERVISORD_CONF`. This configuration file includes settings for a UNIX http server, supervisorctl, an RPC interface for supervisor, and supervisord itself. Additionally, the function ensures the existence of base log directories and logs the creation of the configuration file using `hps_log`.

8.4.54 Technical Description

- **Name**: `configure_supervisor_core`
- **Description**: This function creates a supervisord configuration file at a predefined location and ensures the creation of pertinent log directories.
- **Globals**: [`HPS_SERVICE_CONFIG_DIR`: The directory where the supervisord configuration file is to be created, `HPS_LOG_DIR`: The root directory where logs are to be stored]
- **Arguments**: None.
- **Outputs**: Logs the creation of the supervisord configuration file.
- **Returns**: This function does not return a specific outcome; it performs actions (writes a configuration file) and candidates for several side effects (such as creating directories and writing logs)
- **Example usage**:

```
configure_supervisor_core
```

8.4.55 Quality and Security Recommendations

1. The username and password for the UNIX http server and supervisorctl are hard-coded to 'admin' and 'ignored-but-needed' respectively. It would be advisable to secure this further by either requesting these as inputs from the user or generating them securely within the function.
2. Logs generated by `hps_log` potentially may have confidential or sensitive information, such as the paths of log directories or supervision configuration files. Ensure that they are appropriately protected.
3. The function expects certain global variables (e.g., `HPS_SERVICE_CONFIG_DIR`, `HPS_LOG_DIR`) to be set beforehand. If not set, the function might produce unexpected results. Appropriate error handling or default value assignment could be used to mitigate this.
4. The function runs with root user privileges as specified in the `user=root` configuration setting. This can bring security risks if the supervisord process is compromised. Consider using a non-root user where possible.

8.4.56 `configure_supervisor_services`

Contained in `lib/functions.d/configure-supervisor.sh`

Function signature: 94e779d35f9a3f9fea90f859607c3903523d66e35f08f1be883ff3aeb13f8ede

8.4.57 Function Overview

The `configure_supervisor_services` function acts as a configuration manager for supervisor services in Bash. The function takes a supervisord configuration file, located in the `HPS_SERVICE_CONFIG_DIR` directory, and prepares it for desired supervisor services.

8.4.58 Technical Description

Function: configure_supervisor_services

- **Description**: This function is mainly purposed with the responsibility of supervisor service configuration manipulations in Bash.
- **Globals**: `HPS_SERVICE_CONFIG_DIR` - a global variable that specifies the directory path where the supervisord configuration file is located.
- **Arguments**: This function does not take any user-defined arguments and mainly operates using asserted global variables.
- **Outputs**: Adjustments of the indicated supervisord configuration file, particularized by the declared global variable.

- **Returns**: In typical scenarios, does not specifically return a single variable or object. This function rather operates by enforcing alterations on the indicated supervisord configuration file.
- **Example Usage**: While this process could generally be automated and as such hidden from disconnected user encounters, this function might be initiated as `configure_supervisor_services`, providing the `HPS_SERVICE_CONFIG_DIR` global variable is defined beforehand.

8.4.59 Quality and Security Recommendations

1. This function does not input or output sensitive data, which reduces risk, but as a good practice, any function interacting with files should check for file permissions.
2. Adding error handlers for potential file access issues (e.g., file doesn't exist, or lack of the necessary permissions) would improve the quality.
3. To ensure script interruption upon errors, consider enabling the `set -e` parameter.
4. Make sure that the value of the `HPS_SERVICE_CONFIG_DIR` global variable is set in a secure way, without exposing potential system vulnerabilities.
5. Although they don't specifically apply to this function, always sanitize inputs and ensure least privilege when dealing with files containing sensitive information.

8.4.60 `create_config_dnsmasq`

Contained in `lib/functions.d/create_config_dnsmasq.sh`

Function signature: 5c19d2840751f093e5d235e25b027f015ff8c3397bf8e3295c8f9678f9127911

8.4.61 Function Overview

The Bash function `create_config_dnsmasq` is used for creating and configuring a dnsmasq configuration file. The dnsmasq service is a lightweight network infrastructure tool that provides DNS, DHCP, router advertisement and network boot functions for small networks. The function sources an active cluster filename, sets the configuration file path, checks for the presence of a DHCP IP and generates the necessary settings for the dnsmasq configuration file including PXE/TFTP/DHCP setup, binding specifications, DHCP range generation, DNS configurations, and optional logging and PXE-specific options. If the DHCP IP is not available, an error message is displayed and the function terminates.

8.4.62 Technical Description

- **name:** create_config_dnsmasq
- **description:** Creates and configures the dnsmasq configuration file based on sourced data from an active cluster and defined global variables.

- **globals:** [HPS_SERVICE_CONFIG_DIR: used to define the directory path for the service configuration files, DHCP_IP: used for DHCP IP, DNS_DOMAIN: used for DNS configuration, HPS_TFTP_DIR: used to set tftp root directory, NETWORK_CIDR: used in calculating dhcp-range, DHCP_IFACE: Interface to bind to within the container]
- **arguments:** No arguments used directly by the function
- **outputs:** Logs whether or not the dnsmasq configuration file was successfully generated.
- **returns:** Nothing. Halts execution with exit 0 in case of an absence of DHCP IP.
- **example usage:**

```
create_config_dnsmasq
```

8.4.63 Quality and Security Recommendations

1. Make sure to sanitise all external inputs to the function to avoid injection attacks.
2. Validate all arguments and parameters used within the function to ensure they are in the correct format and type.
3. Use `set -u` to prevent the script from running if undefined variables are encountered, enhancing stability.
4. Consider breaking down this function into smaller functions for better maintainability and troubleshooting.
5. Implement logging mechanisms to track and identify potential issues during execution.
6. Consolidate the use of echo for error outputs to a central error handling mechanism.
7. Consider iterating over a list of required variables (DHCP_IP etc.) at the start of the function, exiting if any are undefined or empty to ensure the function has all necessary information to proceed.

8.4.64 `create_config_opensvc`

Contained in `lib/functions.d/create_config_opensvc.sh`

Function signature: df041d5c7b1d04692533d6a11fc6d3c9e086ce43b936c6d67dad802076e251e5

8.4.65 Function Overview

The function `create_config_opensvc()` is used for creating and configuring OpenSVC agent settings. The function processes the role of a given IP address and generates a configuration for the OpenSVC service. This function creates the necessary directories for configuration and logging, generates temporary files, performs backup of existing configuration files, performs decision logic based on the generated configuration, and writes the authorization key for agent startup. If the key is not provided by the user, the function automatically generates a fallback key.

8.4.66 Technical description

- **Name**: `create_config_opensvc()`
- **Description**: This function is designed to create and initialize configuration files for OpenSVC services.
- **Globals**: `[ conf_dir: Directory for configuration files, conf_file: Specific configuration file, log_dir: Directory for logs, var_dir: Specific variable directory, key_file: File for storing authorization key ]`
- **Arguments**: `[ $1: Used for specifying the role for the IP, if not provided, it is left empty ]`
- **Outputs**: Messages such as 'OpenSVC Configuration Generation Failed' when function fails to generate config, and successful creation of configuration and key files are main outputs.
- **Returns**: Returns `1` if either temp file generation or opensvc_conf generation fails.
- **Example usage**: `create_config_opensvc "node"`

8.4.67 Quality and Security Recommendations

1. Make the script's error messages more verbose for better troubleshooting.
2. Use secure random number generation function or library for generating keys.
3. Check all user given input for injection attacks. The function currently only takes inputs through variables, but if it is ever changed to take user input it is a point of consideration.
4. Implement try-catch blocks (similar functionality can be gained using if checks in bash as done in this function) to handle errors and maintain the code more easily.
5. Improve the way keys are stored and handled. Use of better sophisticated encryption algorithms may be recommended.

8.4.68 `create_iscsi_target`

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: 31b1d82c8fe3715f33280abbecda64305c1dccc58d1fc70c3df63585ebba7d1e

8.4.69 Function Overview

The `create_iscsi_target` is a bash function primarily used for creating an iSCSI target on a remote host. iSCSI (Internet Small Computer System Interface) is an Internet Protocol-based storage networking standard for linking data storage facilities. The function receives three parameters: remote hosts, IP (default is 0.0.0.0), and port (default is 3260). It uses these to create an iSCSI target which exposes block-level storage for use by other networked servers, leveraging the `targetcli` command to perform the actual operations. It also checks for required iSCSI prerequisites before proceeding.

8.4.70 Technical Description

- **Name:** create_iscsi_target
- **Description:** Creates an iSCSI target at the specified remote host using the provided IP and port.
- **Globals:** None
- **Arguments:**
  - `$1: remote_host` - The hostname of the remote machines
  - `$2: ip` - The IP to create the iSCSI target on. Default is 0.0.0.0.
  - `$3: port` - The port to use for the iSCSI target. Default is 3260.
- **Outputs:** Prints out messages about the status of the iSCSI target creation.
- **Returns:** 1 if the remote host is not specified or iSCSI target prerequisites are not met; otherwise, the exit status of the last command executed.
- **Example usage:** `create_iscsi_target my-host 192.168.1.100 3260`

8.4.71 Quality and Security Recommendations

1. Since this function does privileged operations, ensure that it's only run by authorized users or services. Implement proper authorization checks to prevent unwanted usage.
2. Validate input parameters to ensure they are of correct format and within expected ranges. This helps avoid possible command injection vulnerabilities.
3. Handle errors and exceptions appropriately. Right now, if anything fails, the function would just continue with the remaining commands which might lead to inconsistent results.
4. Consider the implications of not using any encryption in your iSCSI traffic. Use IPSec or similarly secure tunneling protocols if this traffic traverses untrusted networks.
5. Consider limiting access to the iSCSI targets from trusted initiator IPs only, to prevent unauthorized accesses.

8.4.72 `create_supervisor_services_config`

Contained in `lib/functions.d/configure-supervisor.sh`

Function signature: 000144298c724dfbd327b7354be11dd901d1726a3f593104f6a6bbdb96329588

8.4.73 Function Overview

This function, `create_supervisor_services_config`, is responsible for invoking three other functions namely: `create_config_nginx`, `create_config_dnsmasq`, and `create_config_opensvc`. It appears to generate configurations for three different services: Nginx, Dnsmasq, and OpenSVC. The exact nature of these configurations will depend on the implementation details of the respective functions.

8.4.74 Technical Description

- **Name:** `create_supervisor_services_config`
- **Description:** This function initiates the creation of configuration for three different services (Nginx, Dnsmasq, Opensvc) by calling their respective functions.
- **Globals:** None used in this function
- **Arguments:** None required for this function
- **Outputs:** Dependent on the outcome of the called functions `create_config_nginx`, `create_config_dnsmasq`, `create_config_opensvc`.
- **Returns:** Nothing explicitly returned but the success of the function depends on the successful execution of the called functions which create the configurations.
- **Example usage:**

```
create_supervisor_services_config
```

This will run each of the three configuration creation functions without any arguments.

8.4.75 Quality and Security Recommendations

1. Cross-check the structure: It would be helpful to ensure that the three configuration generating functions have been defined before this function is called.
2. Incorporating error handling: Each of the called functions should ideally include error handling capabilities to manage any issues that could occur during their execution.
3. Enhance understanding with comments: Commenting your code will enhance the understanding of how each of the called functions operate.
4. Monitor global variables: Although this function doesn't use any, the functions it calls might alter global variables. Using global variables can make debugging difficult and they should therefore be used sparingly.
5. Validate function outcomes: The function could be improved by validating the outcome of each function call. If any call fails, it should either terminate with an error message or attempt to resolve the issue before proceeding.

8.4.76 `detect_call_context`

Contained in `lib/functions.d/system-functions.sh`

Function signature: f167df149452fd670d9f40bd87d52442f2f5d5153026bdfcab5f9c60997d7f96

8.4.77 Function Overview

The bash function `detect_call_context` is designed to identify and echo the current context in which the script is being executed. It handles three main contexts: when the script is sourced instead of directly executed, when it gets invoked as a common gateway interface (CGI), and finally, when it gets directly executed in a shell or reading from stdin without CGI environment variables. If none of these contexts apply, the function defaults to "SCRIPT".

8.4.78 Technical Description

- **Name:** `detect_call_context`
- **Description:** This function identifies the context in which the script is running, which could be either "SOURCED", "CGI", or "SCRIPT". It prints out the current context and then returns.
- **Globals:** [BASH_SOURCE[0]: Describes the source of the bash script, GATEWAY_INTERFACE: A required variable to detect CGI, REQUEST_METHOD: Another required variable to detect CGI, PS1: Helps in explicitly detecting the script]
- **Arguments:** None
- **Outputs:** Prints one of the four possible states - "SOURCED", "CGI", "SCRIPT", or a fallback "SCRIPT".
- **Returns:** `null`
- **Example Usage:**

  ```
  source your_script.sh
  detect_call_context
  ```

  This script would output "SOURCED" if `your_script.sh` contains a call to this function.

8.4.79 Quality and Security Recommendations

1. It is essential that global variables are well-defined and adequately protected in a function. Use local variables or provide default values to prevent empty or undefined global variables issues.
2. Bash lacks some advanced features like well-defined namespaces, classes, or functions. For better security and efficiency, consider using a more powerful scripting language like Python or Perl for complex scripts.
3. Make sure that the script running the function has appropriate permissions. Bash scripts can be a significant security risk if they are writable by any user. Therefore, secure your script by limiting access.
4. Implement error handling and fallbacks for unexpected behaviour or exceptions.
5. The function implicitly trusts that certain global environment variables are not maliciously set. Ensure that these environment variables are validated before use.
6. Regularly update your system and software to prevent security vulnerabilities.

8.4.80 `detect_client_type`

Contained in `lib/functions.d/network-functions.sh`

Function signature: a5996dd77379049ae4564d5c7eaab09a5189d239c610eea12c0d452cd96097e7

8.4.81 Function Overview

The function `detect_client_type()` is designed to determine the type of client making a request and echo it back. The function looks at both the query string and the user agent to make this determination. If the client type cannot be determined, it echoes back "unknown".

8.4.82 Technical Description

- **Name:** `detect_client_type()`
- **Description:** Determines the type of the client from `$QUERY_STRING` or `$HTTP_USER_AGENT`. This function echoes the client type: 'ipxe', 'cli', 'browser', 'script' or 'unknown' if it can't determine the client type.
- **Globals:**
  - `QUERY_STRING`: Utilized to determine the client type based on the presence of certain keywords. If not set, default value is an empty string.
  - `HTTP_USER_AGENT`: Utilized to determine the client type based on the presence of certain keywords. If not set, default value is an empty string.
- **Arguments:** The function does not accept any arguments.
- **Outputs:** Echoes the type of client making the request.
- **Returns:** Always returns 0 as the function is designed to not fail, falling back to a default output of "unknown" when necessary.
- **Example Usage:**

```
$ export QUERY_STRING="via=cli"
$ detect_client_type
cli
```

8.4.83 Quality and Security Recommendations

1. To reduce potential errors or misuse, explicitly document the environments and contexts in which this function should be used or not used.
2. Consider adding error handling for undesired or unexpected input to improve stability.
3. Make sure to sanitize user-generated inputs such as query strings to prevent injection attacks.
4. If feasible, add type checks for input values in cases where the function starts accepting arguments.

5. To prevent leakage of potentially sensitive information, use discretion with echoing data, especially if it includes data from HTTP headers in a web server environment.

8.4.84 `disks_free_list_simple`

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: 23bd3ac7877a7c90a934c9f11fb7a056bf39f675410585baaaa84e107157944c

8.4.85 Function overview

The function `disks_free_list_simple()` is a Bash script that lists all available block storage devices in a system. It singles out regular, non-removable hard drives, discounting specific types of block devices such as loop, ram, md, dm devices, or those with partitions. Moreover, the function ignores devices that are mounted, belong to LVM2, Linux RAID, ZFS storage pool, or exist in a zpool. Whenever possible, it prefers World Wide Name (WWN) identifier for device representation.

8.4.86 Technical description

Below is a technical block describing the function:

- Name: `disks_free_list_simple()`
- Description: This function scans and lists the available block storage devices, taking into consideration specific exclusions.
- Globals: [None]
- Arguments: [No arguments are used with this function]
- Outputs: The function will output a list of free storage devices based on the criteria and filters it applies.
- Returns: No value returned.
- Example usage: After defining the function in bash, simply call `disks_free_list_simple` to use. Note this should be used in a Bash environment such as a script or the command line.

8.4.87 Quality and security recommendations

1. Ensure that you have proper permissions before running this function. It requires access to low-level disk information which might be restricted.
2. Extend error-handling mechanisms to include a fallback or feedback in case the function is not able to execute the commands correctly.
3. Validate that the output of each command is as expected, dealing particularly with edge cases where there might be unique devices or mounting schemes at play.
4. Carefully consider the security implications of exposing low-level hardware information. Apply necessary restrictions when and where needed, such as on a multi-user system.

5. Implement testing methodologies to maintain the function’s integrity and efficiency.

8.4.88 `enable_iscsi_target_reload_on_boot`

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: c18133b271784d139d5b0a951926f1dac4277b5134a96a54ef074be5dfe03d5d

8.4.89 Function overview

This bash script function, `enable_iscsi_target_reload_on_boot`, checks for the existence of the required iSCSI configuration files, verifies if the `target.service` is installed, and if everything is in place, it enables the iSCSI target to reload its configuration at the system boot. If any of these conditions are not met, it returns an error message and exits with a non-zero status code.

8.4.90 Technical description

- Name: `enable_iscsi_target_reload_on_boot`
- Description: This Bash function checks for the required iSCSI configuration files and services, and if found, enables the iSCSI target configuration to reload at system boot.
- Globals: None.
- Arguments: None.
- Outputs: Success or failure message depending on whether the operations were successful or not.
- Returns: The function will return `1` in case of failure (when required files or services are not found) else it will successfully run without explicit return value.
- Example usage: Simply calling the function without arguments is how it is to be used:

```
enable_iscsi_target_reload_on_boot
```

8.4.91 Quality and Security recommendations

1. Input sanitization: Since the function is not currently handling any inputs, this issue doesn’t exist in the present case. However, if any arguments are introduced in future versions, be sure to sanitize them.
2. Error handling: It is recommended to handle other potential errors and exit conditions as well. The function currently checks for two potential issues, but other problems may occur.
3. Logging: To see the complete progress of function, you should add more echo statements at different stages. This will offer better debugging ability if something goes wrong in future.

4. Security enhancements: Validate the required privileges for running the function. In this case, running the script with necessary privileges is crucial for successful operation.
5. You can use more precise tools instead of grep to prevent false positives when searching for the 'target.service'. Do make sure that such tools are installed though.

8.4.92 `export_dynamic_paths`

Contained in `lib/functions.d/system-functions.sh`

Function signature: a6b2a47e08ed460524c491151fd944d515572f0fe9d26a1a2d5d310ad72b99b5

8.4.93 Function Overview

This Bash function, `export_dynamic_paths`, is designed to set and export paths dynamically within a cluster server environment. It makes use of local cluster names to base its operation while providing an alternative default directory path. This function is significant for managing multiple active clusters and ensuring that the active cluster is properly recognized. The function also sets and exports environment variables representing various paths in the cluster's configuration.

8.4.94 Technical Description

- **Name**: `export_dynamic_paths`
- **Description**: A Bash function designed to set and export cluster configuration paths dynamically using the provided cluster name as a reference. This includes the active cluster, the cluster's configuration directory, and the hosts configuration directory. The function also considers the case where an active cluster has not been specified.
- **Globals**: [`HPS_CLUSTER_CONFIG_BASE_DIR`: This global variable provides the root directory for storing cluster configs. By default, its value is set to /srv/hps-config/clusters]
- **Arguments**:
  - `$1`: This argument is the string value that represents the cluster name. If it is not provided, the function will use the currently active cluster (default to empty string).
- **Outputs**: Outputs a warning message "[x] No active cluster and none specified." to stderr if no active cluster exists and none is specified by user.
- **Returns**: Returns 1 if there is no active cluster and none has been specified by the user, or 0 if execution was successful.
- **Example usage**: `export_dynamic_paths 'cluster_name'`

8.4.95 Quality and Security Recommendations

1. Validate inputs at the start of the function. Be sure that the supplied cluster name does not contain unsafe characters (e.g., slashes, backticks, etc.) that could potentially lead to command or path injection attacks.
2. Handle all error or exceptional scenarios. Improve error handling so that more specific messages are returned based on the failure's nature.
3. Make sure that proper permissions are set for the directories and files involved, especially when the function is handling paths and using these to access potentially sensitive data or system configurations.
4. Incorporate a logging mechanism to trace the function's behavior when debugging is required to aid in future troubleshooting.
5. Use more descriptive variable names for readability and maintenance. Ensure the variable and function names accurately describe their purposes or behavior.

8.4.96 `find_hps_config`

Contained in `lib/functions.sh`

Function signature: 2b1359b9639780889fba67b113809b966b3326fe6600551c16100c71c64f76ef

8.4.97 Function Overview

The `find_hps_config` function finds and returns the location of the HPS (High Performance Storage) configuration file from an array of possible locations. If it finds a valid location, the function prints out the location and ends successfully with a zero-status. If the function fails to find a valid location, it ends with a non-zero status.

8.4.98 Technical Description

- **Name:** `find_hps_config`
- **Description:** The function iterates over the `HPS_CONFIG_LOCATIONS` array and assigns the first non-empty and existent file location to the `found` variable. The function prints out the `found` location, and if it is non-empty, the function is successful and returns 0; otherwise, it ends with a return of 1.
- **Globals:** [`HPS_CONFIG_LOCATIONS`]: An array of possible configuration file locations.
- **Arguments:** *Not applicable in this context*
- **Outputs:** Prints the configuration file's location if available.
- **Returns:** `0` if the function is successful in finding a configuration. Otherwise, it returns 1.
- **Example usage:**

```
HPS_CONFIG_LOCATIONS=("/etc/hps/config" "/usr/local/etc/hps/config")
find_hps_config
```

8.4.99 Quality and Security Recommendations

1. The function might silently fail if the `HPS_CONFIG_LOCATIONS` array is not set with valid locations. Guardrails could be put in place to check if the array is set and non-empty before proceeding.
2. The `HPS_CONFIG_LOCATIONS` array should contain carefully considered locations only. Including directories where a malicious user can create files could lead to security vulnerabilities.
3. When a valid file is found, the function stops looking for further files. A more robust function might also check if the located configuration file contains expected data.
4. Surrounding the function argument "$candidate" with double quotes avoids problems with file names that contain spaces or other special characters.
5. Use `set -e` and `set -u` options at the start of the script for a safer script. The `-e` ensures that the script ends if any command, pipeline, or sub-shell exits with a non-zero status. The `-u` option ensures the script exits if an undefined variable is used.

8.4.100 `generate_dhcp_range_simple`

Contained in `lib/functions.d/network-functions.sh`

Function signature: 0b96ef1d9a801dba7584c3c03ea9f1327da1cd2685d343519ee3ae72aa66ecd8

8.4.101 Function Overview

This function, `generate_dhcp_range_simple()`, is used to generate a DHCP range for a given local network provided in CIDR notation. The range generated is based on the network's gateway IP and its broadcast IP, as well as an optional count parameter that specifies the number of IP addresses in the range.

8.4.102 Technical Description

Function Name: `generate_dhcp_range_simple()`

Description: This function generates a range of IP addresses suitable for use as a DHCP dynamic address pool. The IP range is calculated using the provided local network (in CIDR notation), a gateway IP, and an optional count parameter, which defaults to 20 when not specified.

Globals: None

Arguments: - `$1:` network_cidr (e.g. `192.168.50.0/24`) - `$2:` gateway_ip (e.g. `192.168.50.1`) - `$3:` count (default: `20`)

Outputs: A range of IP addresses for DHCP purposes is generated.

Returns: The function doesn't return anything, it just executes side effects.

Example Usage:

```
generate_dhcp_range_simple "192.168.50.0/24" "192.168.50.1"
```

8.4.103 Quality and Security Recommendations

1. For improved data validation, consider adding checks to ensure the CIDR and gateway IP are valid before the function uses them.
2. To avoid potential information leak, consider muting `ipcalc` command internal verbose logs if not necessary for active debugging.
3. Test the function thoroughly using different network setups and gateway IPs to ensure it functions correctly under different attribute values.
4. Consider integrating a logging system for easier debugging.
5. Make sure the script has appropriate permissions, limiting the exposure of the function to avoid unauthorized use. Remember, script that alters network configuration could be a potential security risk if misused.

8.4.104 `generate_ks`

Contained in `lib/functions.d/kickstart-functions.sh`

Function signature: b973477b6d7653d80151f03150fadcfc0d8f1f85a77f353025a267d5257c8529

8.4.105 Function overview

The function `generate_ks()` handles the generation of a kickstart file, used to automate the installation process of an operating system. This function exports different variables containing configuration parameters needed for the OS installation and finally prepares the installation script for rendering.

8.4.106 Technical description

- **Name:** `generate_ks()`
- **Description:** This function accepts two parameters. It exports different variables like macid, HOST_TYPE, HOST_NAME, etc., used for host configuration. It uses helper functions like `hps_log()`, `cgi_header_plain()`, `host_config()`, `cluster_config()`, and `script_render_template()` to log information, get and set configurations, and render the installation script respectively.
- **Globals:** [macid: first argument, represents MAC ID / Unique Host ID of a host machine, HOST_TYPE: second argument, represents the type of the host machine]
- **Arguments:** [$1: MAC ID / Unique Host ID of a host machine, $2: Type of the host machine]
- **Outputs:** Logs about the function calls and configurations, Along with the final render of the installation script.

- **Returns:** Does not return any value. However, the function could be interrupted and denote failure with a return of 1.
- **Example Usage:**

  ```
  generate_ks "00:0c:29:c0:94:bf" "CentOS"
  ```

8.4.107 Quality and security recommendations

1. Avoid usage of global variables as much as possible to reduce side effects and improve function modularity.
2. Input validation and error handling should be more rigorous. Check for the validity of MAC addresses before using them.
3. Preferably restrict access to root or privileged users to reduce potential security risks.
4. Avoid inlining large scripts. Use a separate file to manage large scripts.
5. Implement a logging mechanism to record errors and important events during installation.
6. If possible, encrypt sensitive data like IP addresses, hostnames, and other network details.

8.4.108 `get_client_mac`

Contained in `lib/functions.d/network-functions.sh`

Function signature: 8cf7b608ec665ff47f16996d542d5d3ed0c9392116e453e8abfcc00eab616973

8.4.109 Function Overview

The function `get_client_mac` is used to extract the MAC address corresponding to a given IP address in a local network. It sends a ping to trigger an ARP update, which is then parsed for the required MAC address. If the initial method does not succeed, it uses the `arp` command as a fallback and returns a normalized MAC address.

8.4.110 Technical Description

- **Name:** get_client_mac
- **Description:** This function uses IP address to get its corresponding MAC address from the Address Resolution Protocol (ARP) cache.
- **Globals:** None
- **Arguments:**
  - `$1`: The IP address of the client.
- **Outputs:** Normalized MAC address related to the IP address if found.
- **Returns:**
  - `1` if the IP address is not valid or MAC address is not found.

  - MAC address corresponding to given IP address in normalized form.
- **Example Usage:**
  - `get_client_mac 192.168.1.1`
  - `MAC_ADDRESS=$(get_client_mac 192.168.1.1)`

8.4.111 Quality and Security Recommendations

1. Ensure the user has required permissions to run the `ping`, `ip neigh` and `arp` commands to avoid permission denied errors.
2. Include error handling for cases where ARP does not contain an entry for the given IP address, emitting appropriate error messages.
3. Consider including a validation check for the normalized MAC address before returning it.
4. Use secure command execution to prevent injection attacks.
5. Consider returning a standardized error value, such as a null address or a specific error string, if the MAC cannot be found.

8.4.112 `_get_existing_zpool_name`

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: cc41e1bc5b39a7a3e09eef4edc9166c5f5783d182d1cd860936367173015ae84

8.4.113 Function Overview

This function, `_get_existing_zpool_name`, is used for obtaining the existing ZFS pool name from a remote host. If the ZFS pool name is present, the script prints the name to stdout and returns a zero exit status. If the ZFS pool name is not found, the function returns a non-zero exit status.

The code processes options (e.g. `strategy`, `mountpoint`, `force`, `dry-run`, `no-defaults`) and calls helper functions like `zpool_name_generate` and `zfs_get_defaults` for specific functionalities. It also checks the ZPOOL_NAME before proceeding, validates the policy rules, generates the pool name and selects a disk based on the strategy.

Subsequently, the script performs the creation of the pool, persists the host variable ZPOOL_NAME and provides appropriate logging and error handling throughout the execution.

8.4.114 Technical Description

- Function name: `_get_existing_zpool_name`
- Description: This function is responsible for obtaining the name of an existing ZFS pool from a remote host.

- Globals:
  - `ZPOOL_NAME`: The name of the ZFS storage pool.
- Arguments:
  - `$1`: Command line arguments and options.
  - `$2`: Value of argument where it applies.
- Output: Prints the ZFS storage pool name to stdout if exists.
- Returns: Exit status `0` when pool name is found, `1` or `2` when there is an error.
- Example Usage: `_get_existing_zpool_name --strategy first --mountpoint /tmp --dry-run`

8.4.115 Quality and Security Recommendations

1. The function could benefit from additional input validation. Checking whether the provided arguments are recognizable before their first usage could prevent unexpected behaviors.
2. The error messages can be improved to be more descriptive. For example, stating precisely why a specific helper function is missing could allow an easier troubleshooting experience.
3. The function relies heavily on other functions (helpers), their availability and their output. As such, this function could be prone to break if any of the helper functions are modified. Robust integration tests could help catch these issues.
4. Security-wise, the script does not seem to sanitize inputs when making system calls. This could potentially lead to command injection vulnerabilities. Performing rigorous input validation and sanitization throughout the script would help mitigate this risk.
5. Consider implementing a more comprehensive logging functionality, such as logging every action and result, which can help in debugging and can be crucial in live environments.
6. If possible, make the function idempotent. The function should give the same output and have the same side effects, regardless of how many times it's run with the same inputs. This would ensure predictable behavior.

8.4.116 `get_external_package_to_repo`

Contained in `lib/functions.d/repo-functions.sh`

Function signature: a00cf7324f6795dbbcefc882d9a0346242d31cdeaef0ed00e85c79720244ebc9

8.4.117 Function overview

The `get_external_package_to_repo` function is designed to download a package from a specified URL and save it to a specified repository path. It performs various checks to ensure that the package URL and repository path are valid, that the repository directory exists, and that the package file has a .rpm extension. It logs various information and error messages during its operation, and it uses the `curl` command to download the package.

8.4.118 Technical description

- **Name**: `get_external_package_to_repo`
- **Description**: This function downloads an RPM package from a specified URL to a specified repository path. It checks for correct usage and the existence of the target directory, and it verifies that the URL points to an RPM file. It logs information about its operation and any errors it encounters.
- **Globals**: None.
- **Arguments**:
  - `$1`: URL from which to download the RPM package.
  - `$2`: Path to a directory that will hold downloaded package.
- **Outputs**: Logs information and error messages to standard output.
- **Returns**:
  - `0` if the package is successfully downloaded
  - `1` if incorrect usage
  - `2` if target directory does not exist
  - `3` if URL does not point to an RPM file
  - `4` if it fails to download package URL.
- **Example usage**: `get_external_package_to_repo "http://example.com/package.rpm" "/path/to/repo"`

8.4.119 Quality and security recommendations

1. Use absolute paths for the repository path to avoid potential confusions or errors with relative paths.
2. Validate the URL more thoroughly. Currently, it only checks if the URL ends with `.rpm`. More comprehensive validation would be beneficial.
3. Use the `-s` (silent) option with `curl` to suppress unnecessary output and only display important information.
4. Consider using a more secure protocol, such as HTTPS, for downloading the package to ensure the integrity and security of the download.
5. Add more detailed logging, including timestamps and the full file path of the downloaded files, to allow easier troubleshooting.
6. Rather than relying on return codes, consider propagating more detailed error information to give invokers more context of failures.

8.4.120 `get_provisioning_node`

Contained in `lib/host-scripts.d/common.d/common.sh`

Function signature: 1bbd682413f1c35f8f147b598c84f60028eea1205d56405e50b38d7d551b5b01

8.4.121 Function overview

The `get_provisioning_node` function uses the built-in commands of `ip` and awk to retrieve the IP address of the default gateway in a Unix system. If successful, it prints the IP address of the default gateway to the standard output. This function is generally used for obtaining the IP needed for system provisioning.

8.4.122 Technical Description

8.4.122.1 Name

- `get_provisioning_node`

8.4.122.2 Description

- The function get_provisioning_node retrieves the IP address of the default gateway in a Unix system.

8.4.122.3 Globals

- None

8.4.122.4 Arguments

- None

8.4.122.5 Outputs

- The IP address of the default gateway.

8.4.122.6 Returns

- The default gateway IP address.

8.4.122.7 Example usage

```
gateway_ip=$(get_provisioning_node)
echo $gateway_ip
```

8.4.123 Quality and Security Recommendations

1. Validation: At its current state, the function does not validate the IP address it retrieves. This could pose a problem if the command fails for some reason. Implementing basic validation checks would improve the quality of the function.

2. Error handling: In case the command fails to obtain the IP address, the function currently offers no method of detecting and managing such failure. It is recommended to implement error handling to improve robustness.
3. Comments: More explanatory comments should be added in the function to improve maintainability and readability of the function.
4. Security: It's a good practice to limit the privileges of the script that uses this function to avoid potential security risks.

8.4.124 `__guard_source`

Contained in `lib/functions.sh`

Function signature: e8beb9b32cabb9e73dba64f7b102ad5f1112590b455a914144bd65e5d3001168

8.4.125 Function overview

The function `__guard_source()` is a utility within Bash scripting that helps in preventing a source file from being included and executed more than once. When calling a script source file that has already been loaded, the function immediately returns to prevent duplicated execution.

8.4.126 Technical description

- **Name:** __guard_source()
- **Description:** This function prevents a source file from being executed multiple times in a Bash script. It achieves this by creating a unique variable based on the filename of the source file being called, checking if it already exists, and returning if it does.
- **Globals:** None
- **Arguments:** None
- **Outputs:** None
- **Returns:** 1 if the source file has already been included, 0 if it is the first time.
- **Example usage:**

  ```
  source ./myscript.sh
  __guard_source
  ```

8.4.127 Quality and security recommendations

1. **Input validation:** The function should validate if the value of `${BASH_SOURCE[1]}` exists and is a valid source file before attempting to process it.

2. **Error handling:** The function should have clear error handling and messaging when the source file does not exist or is not accessible.
3. **Security:** Use of the `declare` keyword can lead to code injections if not properly handled. Make sure the input source file name can't be arbitrarily set by the user.
4. **Documentation:** Make sure to properly comment the code to provide a better understanding of what the function does and how it works.
5. **Testing:** Unit test cases should be written to test the function's behavior under different conditions and with a variety of source file names. This will ensure that the function behaves as expected and will catch any errors or issues that might arise in the future.

8.4.128 `handle_menu_item`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: f322a25e1c769cdfd7c9264b5b662eab195f0951e33ef8741bdebbb8691cae0b

8.4.129 Function Overview

The `handle_menu_item` function is primarily designed to manage ipxe menu functions. It is a switch case function that performs different actions depending on the value of the first argument passed to it. Typical operations include initialization, host installation, unconfiguration, and reboots, among others. This function is critical in initializing and managing the state of different hosts in a cluster environment.

8.4.130 Technical Description

- **Name**: handle_menu_item
- **Description**: This function handles various options selected from the ipxe menu.
- **Globals**: None.
- **Arguments**:
  - `$1(Item)`: Menu item to handle (required).
  - `$2(Mac)`: MAC address of a host (required).
- **Outputs**: Logs and helps manage multiple states of a host including installation, local boot, rescue, and so forth.
- **Returns**: Nothing. This function performs action-based operations only.
- **Example Usage**:

```
handle_menu_item "host_install_menu" "00:11:22:33:44:55"
```

8.4.131 Quality and Security Recommendations

1. **Input Validation**: To improve the function, it is advisable to add input validation. Ensure that `$1` and `$2` are not empty before the script runs. Handle the errors accordingly if the inputs do not adhere to the expected format.

2. **Commenting & Documentation**: Commenting and providing more details about each case in the switch statements would improve the maintainability of the code.
3. **Security**: Depending on your environment, if the `hps_log`, `host_config`, `ipxe_reboot`, `ipxe_host_install_menu`, `ipxe_init`, and `ipxe_show_info` functions manipulate sensitive data, ensure they do it securely to prevent potential security vulnerabilities.
4. **Error Handling**: It is suggested to have robust error handling. For example, in the case where the `$item` is unknown, not only log the information but also handle this exception in a way that won't cause potential disruptions.

These recommendations aim to improve code quality, reliability, and security in essential aspects.

8.4.132 `has_sch_host`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 379b6704d9af414555293ef691b70449d39306ef567e44aabf9ef70f91fa692d

8.4.133 Function Overview

The `has_sch_host` function is designed to check if there is any configuration file in the specified directory (`$HPS_HOST_CONFIG_DIR`) that contains a certain type 'SCH'. It first checks if the specified directory exists. If not, it outputs an error message and returns 1 to indicate an error. If the directory exists, it checks every `.conf` file in it for the presence of 'TYPE=SCH'. If at least one file with 'TYPE=SCH' is found, it returns 0 (True). If no such file is found, it returns 1 (False).

8.4.134 Technical Description

- **name**: `has_sch_host`
- **description**: This function checks if there is any configuration file in the specific directory that contains "TYPE=SCH".
- **globals**:
  - `HPS_HOST_CONFIG_DIR`: Directory to search for configuration files.
- **arguments**: None
- **outputs**: An error message if the specified directory does not exist.
- **returns**: 1 if the specified directory doesn't exist or no file containing "TYPE=SCH" is found. 0 if at least one file containing "TYPE=SCH" is found.
- **example usage**: `if has_sch_host; then echo "SCH host config found"; else echo "SCH host config not found"; fi`

8.4.135 Quality and Security recommendations

1. It would be safer to use an absolute path for `$HPS_HOST_CONFIG_DIR` to avoid ambiguity and confusion.
2. It is recommended to standardize the configuration files' extensions, such as `.conf`, for better searchability.
3. Consider improving the error-handling mechanism to make it more robust, for example by detailing which scenarios result in which errors.
4. Further sanitize the search pattern '(^TYPE=SCH)' to prevent potential command injections.
5. Utilize built-in shell checks to validate that "${HPS_HOST_CONFIG_DIR}" is, in fact, a directory, not a file.

8.4.136 `host_config_delete`

Contained in `lib/functions.d/host-functions.sh`

Function signature: e05b6467ee0958047fe16e2eb5a0519a4ceab375f01a2f8a27e8ee6f35cf7400

8.4.137 1. Function Overview

The function `host_config_delete()` is designed to delete a specified host configuration file. The function accepts the MAC address of a host as an argument. It determines the configuration file corresponding to that MAC address in a predefined directory and deletes it. It logs an informational message if the file was deleted successfully, or a warning message if the file was not found.

8.4.138 2. Technical Description

name: `host_config_delete()`

description: This function deletes the configuration file of a host identified by a specific MAC address.

globals: - `HPS_HOST_CONFIG_DIR`: This is the directory where the host configuration files are stored.

arguments: - `$1`: This is the MAC address of the host whose configuration file is to be deleted.

outputs: Informational or warning logs are output depending on whether the operation is successful or not.

returns: - `0`: if the host configuration file is deleted successfully. - `1`: if the host configuration file was not found.

example usage:

```
host_config_delete "00:11:22:33:44:55"
```

It will delete the configuration file mapped to MAC address "00:11:22:33:44:55".

8.4.139 3. Quality and Security Recommendations

1. Make sure the function correctly handles special characters in the MAC address to avoid unexpected behavior or security issues.
2. The function should check if the 'rm' command succeeded before logging a success message. This would improve the reliability of the logging message.
3. The `HPS_HOST_CONFIG_DIR` directory should have appropriate permissions set to prevent unauthorized access or modifications.
4. It would be beneficial to sanitize inputs to reduce the risk of code injection or Directory Traversal vulnerabilities.
5. To reduce the possibility of any potential logs forgery, it would be wise to incorporate a secure logging mechanism.
6. Lastly, check the existence of the `$HPS_HOST_CONFIG_DIR` directory before performing any operation to prevent any unnecessary errors.

8.4.140 `host_config_exists`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 93698d3265775cdeb0581fb007522cdb74832e2c705812027e67b512cd33964b

8.4.141 Function Overview

The `host_config_exists` function is a Bash function in that checks for the existence of a specific configuration file based on a provided *mac* address. It takes a *mac* address as an argument, and constructs a file path to the configuration file using the *mac* address as the file name and a pre-defined directory as its path. It then uses a conditional statement to check if the file exists, returning a boolean indicating the result.

8.4.142 Technical Description

- **Name:** `host_config_exists`
- **Description:** The function checks for the existence of a specific configuration file that correlates to the provided *mac* address.
- **Globals:**
  - `HPS_HOST_CONFIG_DIR`: This global variable defines the directory where the configuration files are stored.
- **Arguments:**
  - `$1: mac`: This argument is the *mac* address which will be used as the base for configuration file name.
- **Outputs:** This function does not directly output any value.

- **Returns:** The function returns a boolean value which indicates whether the sought configuration file exists.
- **Example Usage:**

```
if host_config_exists "00:11:22:33:44:55"
then
    echo "Configuration file exists."
else
    echo "Configuration file does not exist."
fi
```

8.4.143 Quality and Security Recommendations

1. Make sure the `HPS_HOST_CONFIG_DIR` is always defined and set to a valid directory to prevent the function from searching in the incorrect location, leading to incorrect results.
2. Consider validating the *mac* argument to ensure that it is a valid *mac* address. This could prevent potential errors down the line or potential security vulnerabilities if malicious or incorrectly formatted *mac* addresses are provided.
3. The function should handle or communicate any errors it encounters during execution in a secure manner. Currently, it does not communicate any issues outwardly.
4. To better adhere to Unix philosophy, consider making the function output informative messages or codes to standard error when it encounters problems (e.g. when the directory does not exist).
5. Always remember to mark your variables as local where possible to prevent them from leaking into the global script environment, which can be a security risk and a source of bugs. This function does a good job of this by declaring `mac` and `config_file` as local variables.

8.4.144 `host_config`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 389b01155f4ed6b074bab989a189c3e33a5dfb44e6ade66966a2b1966e95960d

8.4.145 Function Overview

The function `host_config()` is an imperative program in bash scripting language. This function accepts four arguments, namely `mac`, `cmd`, `key`, and `value`. Using these parameters, a configuration file is processed for the host machine identified by the MAC address. The function can execute commands (get, exists, equals, set) on the HOST configuration map, utilizing the `key` and `value` parameters. If the configuration file doesn't exist or if configuration has not been parsed for the current MAC address, the configuration file is first parsed and the `HOST_CONFIG` associative array is updated with key-value pairs from this file.

8.4.146 Technical Description

Name: host_config()

Description: Mainly, this function is for configuring host configuration files which store settings for the host machine identified by the MAC address. Commands to process settings include get, exists, equals, and set. An associative array `HOST_CONFIG` is used for this purpose.

Globals: [HOST_CONFIG: An associative array storing key-value pairs from the host configuration file, HPS_HOST_CONFIG_DIR: Directory where host configuration files are stored]

Arguments: - $1: MAC address identifier for host machine - $2: Command to process host settings. Valid commands are get, exists, equals, set. - $3: Key of setting to process. - $4: (Optional) Value to update the setting with if the ‘set’ command is used.

Outputs: Based on the command: - get: Prints the value of the setting identified by the key - exists: Checks if the key exists. No visual output. - equals: Checks if the setting identified by the key matches the value. - set: Logs info about the new setting and update. - A warning if an invalid command is used.

Returns: - No specific integer return values. For ‘get’, ‘exists’, ‘equals’ commands the function effectively returns 0 (true) if successful and 1 (false) if unsuccessful. - For ‘set’ command no explicit return values are declared. - Returns 2 if invalid command is supplied.

Example Usage:

```
mac_addr="00:11:22:33:44:55"
setting_key="Test_key"

# Returns the value of Test_key
host_config $mac_addr get $setting_key
```

8.4.147 Quality and Security Recommendations

1. Strong input validation: Although the function performs some input validation, it could be improved. For example, MAC address format could be validated.
2. Error handling: Currently, if an invalid command is given, the function simply outputs a message and returns 2. Comprehensive error handling here would improve the robustness of the function.
3. Unknown globals: Globals like `HPS_HOST_CONFIG_DIR` are used in the function without prior validation. It’s good practice to check if these are set before usage.
4. Logging: Use a proper logging utility instead of using command-line echo for important operations. Monitoring could also be implemented for security-sensitive operations.

8.4.148 `host_config_show`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 6105ece190686264b3985f4c2de384ff48edc977456deeb1f70f8a863bb065a8

8.4.149 Function Overview

The `host_config_show` function in Bash is used to read a host configuration file that corresponds to a certain MAC address. If a configuration file doesn't exist for a given MAC address, an informational log message is displayed. If found, the function reads through the file, trims any leading or trailing quotes from each value, escapes any embedded quotes and backslashes, and then echoes each key-value pair.

8.4.150 Technical Description

- **Name:** `host_config_show`
- **Description:** Reads a host configuration file that matches a given MAC address. The function will output key-value pairs from the file, with necessary characters escaped for safety. If no such file exists, it logs an informative message.
- **Globals:**
  - `HPS_HOST_CONFIG_DIR`: This global points to the directory where host configuration files are stored.
  - `hps_log`: This global logs messages based on the application's events.
- **Arguments:**
  - `$1`: This is the MAC address of the device. It's supposed to match with a configuration file within the directory specified by `HPS_HOST_CONFIG_DIR`.
  - `$2`: This argument is not used by the function.
- **Outputs:** Key-value pairs from the configuration file, if it exists. Otherwise, an information log message is output.
- **Returns:** Returns 0- indicating successful execution of the function.
- **Example Usage:** `host_config_show "04:0E:3F:A1:B2:C3"`

8.4.151 Quality and Security Recommendations

1. Implement argument validation: `host_config_show` could fail unexpectedly (or silently) if it receives unexpected data. Implement checks for the MAC address format and whether `HPS_HOST_CONFIG_DIR` is set.
2. Use clearer env var names: `HPS_HOST_CONFIG_DIR` and `hps_log` could be renamed to more descriptive names for better readability of the code.
3. Handle failure condition: Does the user need to know when `host_config_show` can't find a particular MAC address? If so, consider changing the return code from 0 in case a corresponding configuration file doesn't exist for the mac address passed.
4. Sanitize file reading: The `read` command used in the while loop might encounter issues dealing with special character sequences. Use `-r` option to prevent this.

5. Protect against variable shadowing: By using `local` for `mac` and `config_file`, this function avoids shadowing variables in the parent scope. Make sure to follow this good practice in the rest of your code as well.

8.4.152 `host_initialise_config`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 9cbdb13ea8edf79033b1a96edf52ebe54e082910b6a8a403b6b2fd86a4d5b486

8.4.153 Function Overview

The function `host_initialise_config()` is an initial set up tool designed to generate and assign a configuration file for a host, identified by its thus passed MAC address. It ensures that the host configuration directory exists. Then, it sets the state of the host config file to "UNCONFIGURED". Additionally, it logs the initialization of the host configuration.

8.4.154 Technical Description

- **Name**: host_initialise_config
- **Description**: This function initialises host configuration file within HPS_HOST_CONFIG_DIR. It takes in the MAC address of the host, creates a configuration file uniquely associated with the host, sets the initial state to "UNCONFUGURED", and logs the initialization.
- **Globals**: [{ HPS_HOST_CONFIG_DIR: The directory to store host configuration files }]
- **Arguments**: [{ $1: MAC address of the host }]
- **Outputs**: Logs the initialization process with the file name
- **Returns**: N/A
- **Example Usage**:

```
host_initialise_config "00:0a:95:9d:68:10"
```

8.4.155 Quality and Security Recommendations

1. The function should include error handling for potential failures while creating directories or setting the state.
2. Consider integrating data validation procedures to ensure that the MAC address provided as an input matches the anticipated format.
3. Remnants of deprecated operations should be removed from the function to prevent future confusion or unintentional uncommenting.

4. To prevent unexpected errors or unauthorized manipulation, it is suggested to place appropriate file and directory permissions on the created configuration file and directory.
5. It would be more secure to generate unique names for configuration files rather than using the MAC address, which can be predictable or easily available. This would make them less susceptible to targeted attacks.

8.4.156 `host_network_configure`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 2d391e7910d04552e8ca1284295323649c38c44df9d4c9f8e335c080e4d38dac

8.4.157 Function overview

The `host_network_configure` is a Bash function that configures network settings for a given host. It accepts the MAC address and host type as arguments, and uses these variables to customize the network setup.

8.4.158 Technical description

Name: host_network_configure

Description: The function takes in the identifier and type of the host, retrieves the Dynamic Host Configuration Protocol (DHCP) IP and CIDR block from the cluster configuration, and configures the network settings accordingly. If the DHCP IP or CIDR block is not present, it logs a debug message and returns 1. It validates the presence of the ipcalc tool required for the configuration process, and if not present, logs another debug message and returns 1. The function calculates the netmask and network base using the ipcalc tool and stores them in local variables.

Globals: - `VAR: dhcp_ip, dhcp_cidr` The DHCP IP address and CIDR block respectively, retrieved from the cluster configuration.

Arguments: - `$1: macid` The MAC address of the host. - `$2: hosttype` The type of the host.

Outputs: Debug messages logged when DHCP IP or CIDR is missing or ipcalc is not installed.

Returns: 1 if either DHCP IP or CIDR block is missing from the cluster configuration, or if the ipcalc tool is not installed.

Example Usage: `host_network_configure MAC_ADDRESS HOST_TYPE`

8.4.159 Quality and security recommendations

1. Make sure to keep sensitive network details in secured and protected configurations.
2. Regularly check the status and availability of `ipcalc` tool.
3. Handle the failure case with a meaningful error message instead of simply logging a debug message.
4. Quaternion further error handling for unexpected return values from the `ipcalc`.
5. Check the validity of the received arguments (MAC address and host type).
6. The function should consider validating the retrieved DHCP IP and CIDR block before proceeding with the rest of the function execution.

8.4.160 `hps_log`

Contained in `lib/functions.d/hps_log.sh`

Function signature: 7097e5f6dbf7eb35af6d5c1fe86b7ea40ce7ed8d95259fe409cfc4e8eb117559

8.4.161 Function overview

The `hps_log` function is a custom log function that logs system events in a file named `hps-system.log` located in the directory specified by `HPS_LOG_DIR`. The function creates logs with timestamps and a log level which can be set from the command input. It also helps organize logs by supporting an identifier input. Lastly, if this function is called without the `HPS_LOG_IDENT` variable having been set, it defaults the identifier as 'hps'.

8.4.162 Technical description

- **name**: hps_log
- **description**: This function logs data with the specified level, message, and identity into a file defined by environment variable `HPS_LOG_DIR`. Default identity is 'hps' when `HPS_LOG_IDENT` is not set.
- **globals**: [HPS_LOG_DIR: Directory to store the log files, HPS_LOG_IDENT: Identity for the logs]
- **arguments**: [$1: Log level, $… : Log message]
- **outputs**: Writes logs to `hps-system.log` file in the location specified by `HPS_LOG_DIR`.
- **returns**: Not applicable since logging functions typically do not return a value.
- **example usage**: `hps_log "error" "This is a sample error message"`

8.4.163 Quality and security recommendations

1. Protect the log file by setting appropriate permissions to prevent unauthorized access or tampering.
2. Regularly rotate and archive log files to avoid them becoming too large.
3. Implement checks to ensure that `HPS_LOG_DIR` is set to a valid directory.
4. Provide error handling mechanisms if the log file cannot be written to.
5. Assign a default value to `HPS_LOG_DIR` that points to a secure and writeable directory if it has not been set.
6. Avoid logging sensitive information to maintain user data privacy.

8.4.164 `hps_origin_tag`

Contained in `lib/functions.d/system-functions.sh`

Function signature: 7cb345cab5af5348e693c4093a5262f82a902f7c5455f50302572d52570efcaa

8.4.165 Function overview

The **hps_origin_tag** function is designed to generate an origin tag based on the context of where it is called. The function primarily uses the following information to construct this origin tag: - Optional override value passed to the function - Current PID, username and hostname if running in a TTY environment - Client IP or MAC address, if available and running in a non-TTY environment - Current PID, if running in a non-TTY environment with no available IP or MAC

This function may be primarily used for logging or tracking the origin of different processes or actions within a system.

8.4.166 Technical description

The technical specifications for this function are:

- **Name**: hps_origin_tag
- **Description**: This function generates an origin tag using several pieces of contextual data, such as the process ID, user, host, and optionally, the client MAC or IP address.
- **Globals**: None directly, the function uses the system-defined $HOSTNAME and $REMOTE_ADDR variables if available.
- **Arguments**: $1 (optional) - An override value that, if provided, will be used as the tag instead of any other checks.
- **Outputs**: Prints a string to stdout. This string will be a 'pid', 'user:host', 'mac', or 'IP' tag based on the available data and presence of a TTY environment.
- **Returns**: Always 0 (success)

- **Example usage**: `bash origin_tag=$(hps_origin_tag "override") # Uses the provided override origin_tag=$(hps_origin_tag) # Generates a tag based on context`

8.4.167 Quality and security recommendations

1. **Sanitize inputs**: The function currently does not perform any checks or sanitization on the input override string. Potentially, this could lead to unexpected behavior if the override contains special characters or whitespace.
2. **Check for command availability**: The function assumes that `id`, `hostname` and similar commands will be available, and only executes them in a subshell if they are allowed to fail. It's recommended to explicitly check for the availability of these commands before proceeding.
3. **Error checking**: The function currently suppresses all command error outputs. Although it handles the most common issues, it should also consider checking and handling other types of errors (e.g., if `printf` fails for some reason).
4. **Unset variables**: Although the function handles unset variables gracefully with the `${VAR:-}` notation, it does not explicitly check if `$HOSTNAME` or `$REMOTE_ADDR` are set before trying to use them, potentially leading to subtle bugs. It's suggested to add these checks.
5. **Security of Information displayed**: The function shares information like the user and host in the tags it generates. Be aware that sharing such details can be a security concern, depending on how and where the function is used.

8.4.168 `hps_services_restart`

Contained in `lib/functions.d/system-functions.sh`

Function signature: c09926a993a6d161495b474383d7d4499d1f392527cb63a9bc4fbff3a73c0eba

8.4.169 Function Overview

This function, `hps_services_restart`, is responsible for restarting all services under the supervision of the Supervisor program. The process involves four steps:

1. Configuring the services which require supervision.
2. Creating the necessary configuration file for those services.
3. Reloading current Supervisor configurations to apply any changes.
4. Using `supervisorctl`, a control tool for Supervisor, with a configuration file located at `HPS_SERVICE_CONFIG_DIR/supervisord.conf` to restart all services.

This function is particularly useful when you have made changes to your services or updated them and need to restart them for those changes to take effect.

8.4.170 Technical Description

- **Name:** hps_services_restart
- **Description:** This function configures, creates the config, reloads the config, and restarts all Supervisor services.
- **Globals:** [HPS_SERVICE_CONFIG_DIR: The directory where the supervisord configuration file is located]
- **Arguments:** No arguments are expected.
- **Outputs:** This function does not produce any notable output beyond potential standard output and errors from Supervisor.
- **Returns:** By default, if successful, this function will not return anything. However, if an error occurs during execution, it will return the error message.
- **Example Usage:** `hps_services_restart`

8.4.171 Quality and Security Recommendations

1. Consider improving error handling to allow for easy debugging. Address every potential point of failure within the function, such as failing to configure services, not being able to create or reload the configuration, and not being able to restart a service.
2. Environment variables like `HPS_SERVICE_CONFIG_DIR` should be properly isolated and validated to prevent injection attacks.
3. The function does not currently validate the configuration file (`supervisord.conf`). A corrupted or improperly formatted file could lead to undefined behavior or interrupt running services.
4. Document more thoroughly about the dependencies ("configure_supervisor_services", "create_supervisor_services_config", "reload_supervisor_config", and "supervisorctl"). Ensure they are securely coded and follow best practices.
5. Ensure that the user running this function has the necessary permissions to restart these services. Accidentally running this function as an unauthorized user can cause the services to stop working.

8.4.172 `hps_services_start`

Contained in `lib/functions.d/system-functions.sh`

Function signature: d803223c5cd9a326d513c4b57b5085266767da7cb4e9c6c99acebea677274834

8.4.173 Function overview

The `hps_services_start` function is meant to start all services under supervisor control. It first configures the supervisor services, then reloads the supervisor configuration.

After that, it uses the `supervisorctl` command to start all the services defined in the supervisord configuration file.

8.4.174 Technical description

Here is a definition block for `hps_services_start` function.

- Name: hps_services_start
- Description: This function starts all services under supervisor control.
- Globals: [HPS_SERVICE_CONFIG_DIR: The directory that includes the supervisord configuration file]
- Arguments: None
- Outputs: It does not return any output as it interacts directly with the supervisor control manager to configure and start services.
- Returns: The status code of the `supervisorctl start all` command.
- Example Usage:

```
hps_services_start
```

8.4.175 Quality and security recommendations

Here are a few improvements that could be made:

1. Add error handling: The function does not handle any potential errors that might occur during starting the services. Proper error handling can be implemented to return useful error messages and halt the script execution when necessary.
2. Validate variable: The variable `HPS_SERVICE_CONFIG_DIR` should be validated before passing it to the `supervisorctl` command to prevent potential command injection attacks.
3. Add Documentation: Each step in the function could use commenting explaining what it does in detail. This would help future developers understand the function more easily.
4. Status Check: After executing the `supervisorctl` command, the function should check and confirm whether the services have indeed been started or not.

8.4.176 `hps_services_stop`

Contained in `lib/functions.d/system-functions.sh`

Function signature: 7353ea08d49309b1aa8e3187b443507d430fd727f57e8ecc29accbe8383b6169

8.4.177 Function Overview

The `hps_services_stop` function is a simple bash function in a script that uses the `supervisorctl` command to stop all currently running services under supervision. The services are managed by supervisord, a supervisor process control system. The function works by passing the configuration file located at `$HPS_SERVICE_CONFIG_DIR/supervisord.conf` to `supervisorctl` command to control the services.

8.4.178 Technical Description

- **Name:** `hps_services_stop`
- **Description:** This function uses the `supervisorctl` command to stop all currently running services which are being managed by supervisord, a supervisor process control system.
- **Globals:**
  - `HPS_SERVICE_CONFIG_DIR`: This is the directory for the supervisord configuration file
- **Arguments:** There are no arguments needed for this function.
- **Outputs:** The function outputs the resulting status of stopped services on the terminal
- **Returns:** No specific return value, it only executes the command within its body.
- **Example usage:** `bash  hps_services_stop`

8.4.179 Quality and Security Recommendations

1. Check that `HPS_SERVICE_CONFIG_DIR` is set and exists before using it, to prevent any command injection or path traversal issues.
2. Implement error checking, for example checking whether the `supervisorctl` command was successful or not, and handle any failures gracefully.
3. Avoid running this function with root privileges unless necessary as it can stop all services, which could have unwanted effects.
4. Validate and sanitize all shell inputs, in this case the `HPS_SERVICE_CONFIG_DIR`, to enforce a tighter security standard and prevent processing unexpected or harmful input.
5. It would be beneficial to log the status messages from supervisorctl for auditing purposes.

8.4.180 `initialise_distro_string`

Contained in `lib/functions.d/configure-remote-host.sh`

Function signature: c440af9a86acddde30570b73ae6d52c7bddf38d765f513ea26ebf15ee4805200

8.4.181 Function overview

The `initialise_distro_string` function is a Bash function that creates a string giving a basic description of the system's distribution. It works by checking for the system's architecture, manufacturer, operating system, and version. If it can't find the OS name and OS version, it will list them as "unknown."

8.4.182 Technical description

- **Name:** initialise_distro_string
- **Description:** This function generates a string containing a description of the system's distribution. It takes no arguments and will list "unknown" if it cannot find the OS name and version.
- **Globals:** [None]
- **Arguments:** [None]
- **Outputs:** The function outputs a string in the format "[cpu]-[mfr]-[osname]-[osver]".
- **Returns:** Doesn't return.
- **Example usage:**

```
distro_string = $(initialise_distro_string)
echo ${distro_string}
```

8.4.183 Quality and security recommendations

1. Include error handling: If the `/etc/os-release` file doesn't exist or can't be accessed, it would be prudent to add error handling logic to the function and notify the user.
2. Validate variables before usage: To ensure only expected values are used, add sanity checks for the variables.
3. Use more strict condition checks: The function currently considers any existing `/etc/os-release` file as valid. Not just its presence, but also the correctness of its format should be checked.
4. Document the function: The function would benefit from inline comments explaining the logic, and a block comment giving a brief overview of the function and its usage.
5. Use underscore in function name: Using underscore (_) between words instead of camel case (`camelCase`) make function names more readable in bash script.

8.4.184 `initialise_host_scripts`

Contained in `lib/functions.d/configure-remote-host.sh`

Function signature: f3a19023175709bb6341f62f9bd565ec7430df7cae722dcd18cf2a20800ee478

8.4.185 Function Overview

The `initialise_host_scripts()` function in Bash is used to initialize host scripts. It leverages the `get_provisioning_node` function to define the gateway, gathers the distro string through the `initialise_distro_string` function and then forms a URL to fetch a function bundle. The function bundle fetched is downloaded and stored under "/tmp/host-functions.sh". On successful download, the file is then sourced. In case of a failure in fetching the function bundle, the function returns an error.

8.4.186 Technical Description

- **Name**: `initialise_host_scripts()`
- **Description**: This function is used to initialize host scripts. It fetches the function bundle from a certain URL and downloads it locally. In case of failure in fetching the function, an error is returned.
- **Globals**: [None]
- **Arguments**: [None]
- **Outputs**: This function outputs the status of the operation including the URL from which the function bundle is being fetched, successful or unsuccessful fetching and sourcing of the function bundle.
- **Returns**: This function returns 1 in case of failure in fetching the function bundle.
- **Example usage**: `bash      initialise_host_scripts`

8.4.187 Quality and Security Recommendations

1. Implement a validation check to ensure that the URL is correctly formed and can be accessed. This reduces the risk of script failure due to incorrect URL.
2. Before sourcing the downloaded script, introduce a step to verify its contents. This would protect against downloading and executing malicious content.
3. Consider adding more error handlers in case the functions `get_provisioning_node` and `initialise_distro_string` fail.
4. The destination file path is currently hard-coded which makes the function less flexible. Consider passing the destination file path as an argument.
5. Provide more detailed error messages to improve debugging and user experience.
6. Uses comments for clearer understanding of complexities within the script.

8.4.188 `initialise_opensvc_cluster`

Contained in `lib/host-scripts.d/common.d/opensvc-management.sh`

Function signature: b57d9a20982173cc9b26a4a5f06dac4ca7e6e1eb938901809999cb820013b4c7

8.4.189 Function overview

The `initialise_opensvc_cluster` function is used to set up an OpenSVC cluster. It takes no direct arguments. The function sets up the cluster in a server-side MAC resolution setup, reading configuration values from the HPS system, setting cluster name and node tags, and properly initializing the OpenSVC daemon.

8.4.190 Technical description

- **Name:** initialise_opensvc_cluster
- **Description:** This function uses HPS configs to setup an OpenSVC cluster, set the cluster name and node tags, and initialize OpenSVC daemon.
- **Globals:**
  - **cluster_name:** gets the cluster name from HPS configs
  - **node_tags:** gets the node tags from HPS configs
- **Arguments:** None.
- **Outputs:** Logs messages to the remote log with information and possible issues during the cluster initialization process.
- **Returns:** Return code is 1 if any error occurs and if OpenSVC daemon failed to restart.
- **Example usage:**

```
initialise_opensvc_cluster
```

8.4.191 Quality and security recommendations

1. Consider better error handling: instead of returning a binary (0/1) success/failure status, use different return values for different types of errors to help callers diagnose problems.
2. Validate inputs: As this script uses external HPS configs, ensure that these configuration inputs are valid and safe to protect against configuration errors or potential security risks like command injection.
3. Add comment explanation for the complex commands or steps to enhance the code's readability and maintainability.
4. Use consistent error messaging: Consistently structure error messages to make them easier to understand and to help with troubleshooting.
5. Add detailed logging: To enhance traceability and debugging, log the beginning and end of each operation, as well as any exceptions or unexpected conditions.

8.4.192 `int_to_ip`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 1acb712577aa9213ca920a051e80825c73d7775714d43d10ccad322458fc5946

8.4.193 Function Overview

The function, `int_to_ip()`, translates an integer into its corresponding IP address. Given an IP address represented as an integer, this function will return the dotted-decimal representation of the IP address. Additionally, this function checks from a pool of unused IP addresses and generates a unique IP address not yet assigned to any hosts. Similarly, a unique hostname will also be created. Once an IP and hostname are generated, they will be assigned to the host configuration.

8.4.194 Technical Description

- **Name**: `int_to_ip()`
- **Description**: Translates an IP represented as an integer into its corresponding dotted-decimal format. Also, it assigns IP and hostname to the host configuration after generating unused ones.
- **Globals**:
  - `HPS_HOST_CONFIG_DIR`: Directory of host configuration.
- **Arguments**: `$network_base`: Base integer of entire network.
- **Outputs**: The function outputs the translated IP address from integer to a dotted decimal format.
- **Returns**: It could return 1 if there is no available IP in range or fails to generate a unique hostname.
- **Example usage**:

```
$ int_to_ip 668154256
```

8.4.195 Quality and Security Recommendations

1. Implement proper error-handling to cater scenarios where input passed could not be translated to valid IP address.
2. It is crucial to validate the input integers for their validity in terms of being translatable to IP addresses.
3. Validation check to ensure that the input is indeed an integer and within the expected range.
4. It is recommended to add log entries whenever a new valid IP or hostname is found or whenever an IP or hostname assignment fails.
5. There should be checks to ensure that the created hostnames do not collide with any existing system hostnames.

8.4.196 `ip_to_int`

Contained in `lib/functions.d/host-functions.sh`

Function signature: 7a7e2ac879b38f493155a8d7ebe4e0938b6b76af6dd4451359927f5afb697e52

8.4.197 Function Overview

The `ip_to_int()` is a bash function which takes a single IPv4 address as a string in its standard dotted decimal notation (for example '192.0.2.1'), and converts it into its corresponding 32-bit integer representation.

8.4.198 Technical Description

The following is a detailed breakdown of the function and its components.

- **Name**: `ip_to_int()`
- **Description**: This function parses an IPv4 address and outputs it as a 32-bit integer. It uses bitwise shift operators to move each octet to its correct position in the 32-bit number.
- **Globals**: None used
- **Arguments**: A single argument is expected, `$1`, which represents the IPv4 address to be converted.
- **Outputs**: The function echoes the 32-bit integer representation of the given IP address.
- **Returns**: The function does not specifically return any value, but its command status will be 0 if it executes successfully and a non-zero value if it fails.
- **Example usage**: `ip_to_int '192.168.1.1'`

8.4.199 Quality and Security Recommendations

1. Error Checking: There should be validation to ensure that the IP address provided matches the expected format. If an invalid IP address is input, the function will currently output unexpected or incorrect results.
2. Argument Count: The function should verify that exactly one argument has been provided. If multiple arguments are provided, unexpected output may be returned.
3. Usage of Global Variables: As of now, the function does not have global variables. But if there is a need for them in the future, try to limit the use of global variables as they potentially affect all parts of a program, not just the function itself.
4. Exit Status: The function should provide more expressive exit statuses for different types of errors, such as invalid IP format or incorrect argument number rather than using the default exit status.
5. Command Injection: Although the function does not directly use user input in a command, it still could be vulnerable to command injection if not properly handled. Always sanitize user input.

8.4.200 `ipxe_boot_from_disk`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: f26dc79cd8c9ea270e2758702b63d4a607fd3fd40737c5d947de898af37ff1fe

8.4.201 Function overview

In the Bash shell, the function `ipxe_boot_from_disk` primarily ensures that the system boots from the local disk through BIOS (Basic Input/Output System). This action is accomplished by exiting from the iPXE stack, an open-source network boot firmware. The function first invokes `ipxe_header` then hands control back to BIOS for booting.

8.4.202 Technical description

- **Name**: `ipxe_boot_from_disk`
- **Description**: This function is responsible for facilitating system boot from local disk through BIOS. It achieves this by exiting the iPXE stack. Before handing over the resultant control to the BIOS, it prints a statement notifying of the action and waits for 5 seconds.
- **Globals**: None.
- **Arguments**: None.
- **Outputs**: This function prints a message "Handing back to BIOS to boot" and then initiates a 5-second sleep.
- **Returns**: Doesn't return any value.
- **Example usage**: `ipxe_boot_from_disk`

8.4.203 Quality and security recommendations

1. Always ensure that this function is called in a safe and secure environment. Unauthorized access or modifications could lead to serious system damage or data loss.
2. When calling the method `ipxe_header`, please make sure that it is properly defined and doesn't have any undesired side effects.
3. As it automatically initiates a system boot after 5 seconds, it's necessary to warn the user beforehand.
4. Consider adding error handling code to ensure its successful execution.
5. Verify that the BIOS settings permit booting from the local disk; else the function may not yield the expected results.

8.4.204 `ipxe_boot_installer`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: ad6c7317bbb7a71cfcd8f86037b9440faaacb0b1a309a4e03c807b567e589b21

8.4.205 Function overview

The function `ipxe_boot_installer ()` automates the process of PXE booting installation for new hosts in a network using iPXE (an open-source network boot firmware). It takes in the host type and the desired profile, and then retrieves the necessary parameters such CPU, Manufacturer (MFR), Operating System Name (OSNAME) and Version (OSVER). It also checks whether the new host is already installed, and exits if that is the case. The function currently supports `rockylinux` but has placeholders for `debian`.

8.4.206 Technical description

- name: `ipxe_boot_installer`
- description: This function manages configuration and boot of PXE for a new host installation.
- globals:
  - `$HPS_DISTROS_DIR`: Directory path to the list of supported distributions.
  - `mac`: The MAC address of the host.
  - `CGI_URL`: The URL of the CGI script to generate the kickstart file.
- arguments:
  - `$1: host_type`: Type of the host, e.g. server, workstation.
  - `$2: profile`: The desired profile.
- outputs: The function outputs a string representation of a iPXE boot script.
- returns: Returns nothing. However, in case of error, it terminates and echoes an error message.
- example usage:

  ```
  ipxe_boot_installer workstation minimal
  ```

8.4.207 Quality and security recommendations

1. Make sure that the `HPS_DISTROS_DIR` points to a secure and reliable source for the distributions.
2. Avoid hard-coding URLs and paths inside the function. Instead, pass them as arguments or set them in a separate configuration file.
3. Check that all globally used variables, like `mac` and `CGI_URL`, are safely defined and handled to avoid overwriting or misusage.
4. Add support for other distribution types, such as Debian, to heighten the function’s versatility.
5. Implement proper error handling and logging to identify and resolve issues quickly, contributing to function robustness.
6. Finally, to enhance security, in cases of failure, consider using placeholders for logging error messages that avoid printing sensitive information (like file paths, URLs, or parameter values).

8.4.208 `ipxe_cgi_fail`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: 98eb961ae3ed7dfffc7f4ae68cd1c88aa5f47672ee53b71b29f0428922e8efaa

8.4.209 Function overview

The `ipxe_cgi_fail` function is used within a Bash environment to display a failure message using the IPXE header. This header message is provided through the `$1` parameter. The function logs the error message, displays it to the user, sleeps for 10 seconds, and then initiates a system reboot.

8.4.210 Technical description

- **Name:** `ipxe_cgi_fail`
- **Description:** This function displays an IPXE failure message to the user, logs the error message, waits for 10 seconds, and then reboots the system.
- **Globals:** None.
- **Arguments:**
  `- $1: the error message to be displayed and logged.`
- **Outputs:** Prints the failure message using the IPXE header design. It also echoes the error, waits for 10 seconds, and then initiates a reboot sequence.
- **Returns:** Nothing. After performing its operations, the function terminates Bash script using the `exit` command.
- **Example usage:**

```
ipxe_cgi_fail "Failed to complete operation"
```

8.4.211 Quality and security recommendations

1. Ensure that the input parameter is properly sanitized to prevent potential command injection attacks.
2. Remember to handle the scenario where the `$1` argument may be empty or undefined, as such a situation may lead to unexpected behavior from the function.
3. Considering that the function is forcing a system reboot, ensure that it is only accessible and executable by authorized users to prevent potential misuse.
4. As this function is logging error messages, ensure the logging system itself is secure, and sensitive information is not being inadvertently logged.

8.4.212 `ipxe_configure_main_menu`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: 1f7c5adf70ab3b0d8ce1dc04122eff4ae6c8141a95956ef26ee00221876e111c

8.4.213 Function overview

The `ipxe_configure_main_menu` function presents a main menu to the user for the configuration of the host. It is important when a host is deployed in a cluster but not yet configured. The menu generated includes options for host installation, recovery, system configuration, advanced settings, etc. The function also checks whether forced installation is activated or not and sets up the menu accordingly.

8.4.214 Technical description

Name: `ipxe_configure_main_menu`

Description: This function generates a main configuration menu to be presented in the context of a host system within a cluster. It offers several host and system options, ranging from installation settings to advanced configurations. Forced installation status is also checked and appropriate menu options are set based on the status.

Globals: [`mac` : The MAC address of the host. `FI_MENU` : A string which stores the menu item related to forced installation status. `FUNCNAME` : An array variable containing the function names in the call stack, with the function at the bottom of the stack in the first position. `CGI_URL` : The URL to send CGI requests]

Arguments: - None

Outputs: Delivers main configuration menu

Returns: No return from the function as it is not supposed to yield any particular value

Example usage: ipxe_configure_main_menu

8.4.215 Quality and security recommendations

1. Implement rigorous input validation and sanitization to enhance security and avoid potential command injection attacks.
2. Implement and enhance error handling to cover potential failures or anomalies during the process.
3. Document any other potential global variables being used within this function.
4. Implement disaster recovery host (DRH) recovery functionality as the menu item is yet to be implemented.
5. Encrypt sensitive data such as MAC addresses, URLs, and so on to improve security.
6. Ensure the correct generation and interpretation of the `funcname[1]` array variable to avoid potential errors.
7. Use clear and explanatory log messages to aid in debugging and maintaining the script.

8.4.216 `ipxe_header`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: f724cbc120f1a1eae5cf985238d6bc44a932f4521832fa8959c47118c5068ed5

8.4.217 Function overview

The function `ipxe_header` is meant for initial preboot execution environment setup. This includes setting up a CGI header via the `cgi_header_plain` function, defining several variables to be used in IPXE scripts and outputting an embedded IPXE script.

8.4.218 Technical description

- **Name:** ipxe_header
- **Description:** This function is responsible for creating a Preboot eXecution Environment (PXE) header via a standard CGI header. This ensures there is no boot failure. It also sets up several variables for use in IPXE scripts and outputs an embedded IPXE script.
- **Globals:** [CGI_URL: URL of the boot manager script, TITLE_PREFIX: concatenated string containing cluster name, mac address and network IP]
- **Arguments:** None
- **Outputs:** Outputs an embedded IPXE script which sets the log message, fetches an image for logging, and displays connection and client information to the cluster.
- **Returns:** None
- **Example usage:**

  ```
  ipxe_header
  ```

8.4.219 Quality and security recommendations

1. All global variables should be properly sanitized and their contents validated to avoid potential code injections or manipulation of expected values.
2. Generally avoid crafting URLs manually, when possible, as it opens up potential for URL manipulation and injection attacks.
3. Place detailed and pertinent comments on critical sections of the code to ensure maintainability and understandability.
4. Parameters passed to the function could be validated - by checking type, format or range - before using them inside the function to make sure they are in expected form.
5. Implement error handling mechanism for the cases when “imgfetch” fails or has an error.

8.4.220 `ipxe_host_install_menu`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: 0627b23e1b7a33e58451ad40a8e31ebc0e9911be4bc534e1afb8dc54dea050c4

8.4.221 Function Overview

The function `ipxe_host_install_menu` is a part of the iPXE software. iPXE is an open-source boot firmware that is used to install operating systems over the network. This particular function is used to generate a selectable installation menu wherein users can select a particular installation option (thin compute host, storage cluster host, etc.). Upon selection, the chosen installation instruction is processed, and an appropriate network boot is initiated.

8.4.222 Technical Description

- **Name**: ipxe_host_install_menu
- **Description**: This function generates a menu of installation options for the iPXE software. It does this by using the ipxe_header function, and generating a list of installation options through the use of the `cat` command. Once a selection is made, an appropriate installation process is initiated remotely over the network.
- **Globals**: [TITLE_PREFIX: A variable used to prefix the menu title]
- **Arguments**: None
- **Outputs**: Prints out a menu of installation options.
- **Returns**: Initiates the installation process of the selected option.
- **Example Usage**: `ipxe_host_install_menu`

8.4.223 Quality and Security Recommendations

1. Using globals in bash scripts should be avoided if possible. Globals are not thread-safe, and their values can be changed by any part of the code, making it hard to track bugs.
2. The function lacks error handling. It would be advisable to include checks for possible point of failures like availability of network connection, validity of installation options, etc.
3. If the CGI_URL is created or modified in some other part of the code, ensure all the possible URL values are sanitized to prevent the possibility of code injection attacks.
4. Ensure that the communication between the servers in the network is encrypted to prevent man-in-the-middle attacks during the network boot process.
5. Check that the file paths, networks addresses and other resources the script interacts with are properly secured with correct permissions. This prevents unauthorized access or changes to these resources.

8.4.224 `ipxe_host_install_sch`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: 76d8d63df923bfeaa3d9b18625e12e7909180d679baee564a7d5760de6c84baa

8.4.225 Function overview

The function `ipxe_host_install_sch` is designed for menu presentation purposes in order to install a Storage Cluster Host (SCH). Upon execution, it utilizes the `ipxe_header` function to generate uniform iPXE menu headers. The function then proceeds to print clear statements regarding the utility and consequence of a SCH installation, along with options for types of installation. Ultimately, the data on actions to take (like installing now) and logging messages are fetched and replaced with appropriate URLs from predefined variables. It facilitates the configuration and initiation of SCH installation procedures.

8.4.226 Technical description

- **Name**: `ipxe_host_install_sch`
- **Description**: This function presents a menu for the installation of a Storage Cluster Host (SCH).
- **Globals**: None.
- **Arguments**: None.
- **Outputs**: Prints a menu with storage cluster host installation options, with a section for informational items, back navigation, and two different installation methods.
- **Returns**: None.
- **Example usage**:

```
ipxe_host_install_sch
```

8.4.227 Quality and security recommendations

1. Encapsulate the function body to prevent external interference with the internal state.
2. Use explicit declarations for all internal variables and enforce immutable declarations where possible.
3. Include error handling provisions within this function to ensure that it fails gracefully and presents meaningful error logs for diagnostic purposes.
4. Ensure that the URLs used in the `chain` and `imgfetch` commands contain only trusted and secure links to prevent potential security breaches.
5. Regularly update and review the function to ensure compatibility with future versions of related tools or technologies.

8.4.228 `ipxe_init`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: b99f8e1adfe13d429e91c39da6aeab71b263f99c8d73aa5f35dd7351988760e2

8.4.229 Function Overview

This function, `ipxe_init`, is designed primarily to initialize the iPXE environment during boot, particularly in clusters where identification of specific hosts might not yet be complete due to unascertained MAC addresses. The function sets a configuration URL, fetches and loads the associated configuration file. The ability to handle exceptions where there is no available host configuration is also integrated into the function.

8.4.230 Technical Description

- **Name:** `ipxe_init`
- **Description:** The function calls header file and requests IPXE configuration from a specific URL, fetches the config, loads it, and executes. It has a commented segment of codes for handling situations where there is no configuration file found.
- **Globals:** [CGI_URL: The URL of the server from which the IPXE configuration file will be fetched]
- **Arguments:** There are no arguments.
- **Outputs:** The function doesn't have a return output as it mainly performs operations of fetching, loading, and executing configurations.
- **Returns:** Does not return a value.
- **Example Usage:**

  ```
  ipxe_init
  ```

8.4.231 Quality and Security Recommendations

1. Error Handling: Uncomment and adapt the code section dealing with the absence of a host configuration. It is essential that your system handles errors or exceptions properly to resist crashes or inaccuracies.
2. Input Validation: While the current function does not have any direct user input, if modifications are made in the future to incorporate any, rigorous input validation should be conducted to reduce potential security risks.
3. Code Comments: There are good code comments in the functions, which increases the readability and maintainability of your code. However, ensure that any changes or updates to the code are reflected in the comments as well.
4. Secure Fetch: When fetching files or configurations over a network, ensure that secure protocols are used to prevent man-in-the-middle attacks.
5. Testing: Conduct rigorous testing to ensure the function operates as expected under a variety of conditions.

8.4.232 `ipxe_reboot`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: c2e2fb28eaa8f9898d8b5cb181b2b278c884005d1a98813c38797c3225f12b52

8.4.233 Function Overview

The function `ipxe_reboot()` is used to implement a reboot process for an iPXE server. It takes an optional message as argument, logs the reboot request with that message, prints an iPXE header, displays the provided message if it is defined, waits for 5 seconds, and finally issues a reboot command.

8.4.234 Technical Description

- **name**: `ipxe_reboot`
- **description**: This function allows the iPXE server to reboot. It can display a message, pause for a moment for users to read the message, and then force a system restart.
- **globals**: None.
- **arguments**:
  - `$1: MSG`: An optional argument that holds the message which is momentarily displayed before the system restart.
- **outputs**: If a message is provided, that message will be displayed on the screen. Regardless, "Rebooting…" text will be displayed for 5 seconds right before the reboot.
- **returns**: No return.
- **example usage**:

```
ipxe_reboot "Scheduled maintenance. Machine is going down for
↪ reboot."
```

8.4.235 Quality and Security Recommendations

1. Make sure that the output log files have appropriate file permissions to prevent unauthorized access.
2. Validate the type and format of input MSG before accepting it as a valid argument.
3. Although the echo command is not likely to fall victim to command injection, be cautious of using unfiltered input in other contexts or more complex implementations.
4. Documentation could be included within the function to encourage best practices.
5. Consider adding error handling for the "reboot" command in case it fails to execute for some reason.

8.4.236 `ipxe_show_info`

Contained in `lib/functions.d/ipxe-menu-functions.sh`

Function signature: cbd14427c29a67a77d52cb0fd250b99f45cb94110f4d4b0a3f1123ec4fe219a4

8.4.237 Function overview

The `ipxe_show_info()` function is mainly used to display different configurations of a particular host. This function pulls up information such as the host's IP, hostname, platform, UUID, serial number, product, cluster configuration, and HPS (High Performance Storage) paths.

The function achieves this by taking a single argument, which determines the category of information that will be displayed for the host. The categories include `show_ipxe`, `show_cluster`, `show_host`, and `show_paths`.

8.4.238 Technical description

- **Name**: `ipxe_show_info()`
- **Description**: This function displays specific information about a host, based on what category is passed as an argument.
- **Globals**:
  - `HPS_CLUSTER_CONFIG_DIR`: The directory where the cluster configuration is stored
  - `CGI_URL`: Link to CGI (Common Gateway Interface) script which is used to implement the command `process_menu_item`
  - `HPS_CONFIG`: The file where the HPS (High Performance Storage) configuration is stored
- **Arguments**:
  - `$1: category`: The type of information to display about the host. The options are `show_ipxe`, `show_cluster`, `show_host`, and `show_paths`.
- **Outputs**: Information about the host in the requested category, such as IP address, hostname, platform, UUID, serial and product numbers, and more.
- **Returns**: Does not return a value.
- **Example usage**: `ipxe_show_info show_ipxe`

8.4.239 Quality and security recommendations

1. Always ensure variable sanitization before using any variable in command substitution. This removes potential harmful commands being hidden as a value for a variable.
2. Always quote the variables. This will prevent word splitting and pathname expansion.

3. Implement error handling for when file reading or command chain replacement fails, or when an unknown item category is passed as an argument.
4. Implement checking for edge cases where the category argument is not passed at all.
5. The `ipxe_show_info` function is essentially storing, processing, and displaying sensitive information about a host. It is recommended to add necessary security measures to protect this sensitive data from potential breaches.
6. Use `printf` instead of `echo` for better string handling, especially while displaying file contents.
7. Consider using local variables. This can help in preventing variable clashing and accidental modification of environment variables which can have impact on how the system functions.

8.4.240 `iscsi_targetcli_export`

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: 1febc38504cbf3c2f9e3c9454db03a4906bf3c248c462581bd42ab50e69d0296

8.4.241 Function overview

The `iscsi_targetcli_export()` is a Bash function part of the iSCSI protocol used for block-level storage traffic management between server and client in a network. This function creates a block-based backstore, an iSCSI target, a Logical Unit Number (LUN), sets the iSCSI target parameters, creates the portal listening on the specified IP address and port, and saves the getConfigureduration.

8.4.242 Technical description

- Name: `iscsi_targetcli_export`
- Description: It executes a sequence of targetcli commands to create an iSCSI target with a block-based backstore and certain attributes. The target and portal creation can be conditional based on the new_target flag.
- Globals:
  - None
- Arguments:
  - `$1`: The IQN (iSCSI Qualified Name) for the iSCSI target
  - `$2`: The path to the zvol block device
  - `$3`: Backstore name
  - `$4`: IP address to bind for this iSCSI target
  - `$5`: Port to bind for this iSCSI target
  - `$6`: Flag to indicate new target (Value: 1) or existing target (Value: 0)
- Outputs: None

- Returns: None. However, if an error occurs within `targetcli` commands being executed, the error will be written to stderr by `targetcli`.
- Example usage:

```
iscsi_targetcli_export "iqn.2022-01.com.example:target1"
↪  "/dev/zvol1" "backstore1" "192.168.1.10" "3260" "1"
```

8.4.243 Quality and security recommendations

1. Implement error checking and input validation: Currently, the function doesn't check if the provided arguments are valid. Input validation to ensure valid IQN, Zvol path, IP address, etc. can help mitigate errors and enhance the function reliability.
2. Sanitize all inputs: To prevent any command injection or arbitrary command execution, all inputs should be properly sanitized, especially when they're being directly used in command-string construction as in this function.
3. Handle `targetcli` command errors: The function doesn't handle any errors that may arise during the command execution. Error handling or exit codes from all `targetcli` commands can greatly improve the robustness of the function.
4. Logging: Implement extensive logging to capture all activities and errors. This is crucial for troubleshooting and system auditing.
5. Consider explicit return values: Right now there are no explicit return values. Providing explicit return values (return 0 upon success or a unique non-zero value upon each type of error) would make this function more reusable by other scripts/-functions.

8.4.244 `_is_tty`

Contained in `lib/functions.d/system-functions.sh`

Function signature: 53fbf6cc003bc7d9b4614fc9d372f3471ed01447874f4db99da2816eb3cb7e69

8.4.245 1. Function Overview

The Bash function `_is_tty` is designed to check whether the standard input (stdin), standard output (stdout) or the standard error (stderr) of the current terminal is attached to a tty (terminal).

8.4.246 2. Technical Description

- **Name:** `_is_tty`
- **Description:** The function checks if standard input (stdin), standard output (std-out), and standard error (stderr) are currently attached to a tty (terminal). This is

achieved by using the `-t` test which returns true if the file descriptor is open and associated with a terminal.

- **Globals:** None
- **Arguments:** None
- **Outputs:** No explicit output. Internally the function returns with a status of 0 if any of the standard I/O streams are attached to a tty, or with a non-zero status otherwise.
- **Returns:** It returns true if at least one of stdin, stdout, or stderr is attached to a terminal. Otherwise, it returns false.
- **Example Usage:**

```
if _is_tty; then
    echo "We're in a tty terminal."
else
    echo "We're not in a tty terminal."
fi
```

8.4.247 3. Quality and Security Recommendations

1. Do not use this function if data privacy is your concern. In shared systems, other users can read or write to this terminal if they know its tty device file.
2. Always check the result of the function to handle the non-interactive environments appropriately.
3. It's highly recommended to handle the return status of this function properly to prevent faults in scripts that depend on a tty terminal.
4. Since the function has no arguments or global side-effects, it's safe to use this in any part of your scripts. But be aware that it only checks the state of the terminal at the time the function is called, not continuously.

8.4.248 `load_opensvc_conf`

Contained in `lib/host-scripts.d/common.d/opensvc-management.sh`

Function signature: 34b85a1047a61b533260c1512950ed5d8805b33c625908de3bcc4caf2d262537

8.4.249 Function Overview

This function, `load_opensvc_conf()`, is mainly used to fetch, validate, and load the OpenSVC configuration from a specified gateway. The function performs the following actions:

1. Resolves a provisioning node.
2. Creates a directory for the configuration file if it does not already exist.
3. Downloads and temporarily stores the configuration file from the gateway.

4. Checks the integrity of the fetched configuration file.
5. If the fetched file is unchanged, it discards the temporary file and logs a message stating that there is no need to restart.
6. If the file has changed, it makes a backup of the current configuration file, replaces it with the new one, and restarts the daemon.

8.4.250 Technical Description

- Function: `load_opensvc_conf()`
- Description: The function fetches, checks, and loads a new opensvc conf file from a specified gateway. If the file is new or modified, it backs up the old file, installs the new one, and restarts necessary services.
- Globals: `conf_dir`: The directory where the OpenSVC configuration file is stored,`conf_file`: The OpenSVC configuration file.
- Arguments: None.
- Outputs: Logs a message regarding the process completion status and potential error messages to the corresponding system log.
- Returns: `0` if the function successfully completes all its tasks and `1` if any of the tasks encounter an issue that causes it to terminate.
- Example usage: This function is not intended to be invoked with any command-line arguments and thus its usage is simply: `load_opensvc_conf`.

8.4.251 Quality and Security Recommendations

1. Always ensure the validity of the configuration file source (the “gateway”) by using trusted certificates.
2. Make sure that the user running the script has proper permissions for all the necessary operations, like creating directories, fetching data, and restarting services.
3. It would be beneficial to implement more sophisticated error handling for the various stages of the function.
4. Strengthen the validation checks for the fetched configuration file.
5. When running the `curl` command, use the `-S` or `--show-error` flag to ensure error messages are displayed if an error occurs.
6. Use the `-s` or `--silent` with `curl` to not show progress meter but still show error messages.
7. To protect the original file, make a backup copy before making changes.
8. Encrypt sensitive parts of the configuration file to ensure security.
9. Use detailed log entries for any operation in order to trace back in case of any issue.
10. Always check the return status of each command, rather than assuming it will succeed. This will help to capture and handle any errors.

8.4.252 `load_remote_host_config`

Contained in `lib/host-scripts.d/common.d/common.sh`

Function signature: 7a3ad554ca390fc6c05a7c79d9c40323f388eca53f8e402fc2f7f2adaf358fc0

8.4.253 Function overview

The load_remote_host_config function is used to obtain configuration data for a host from a remote source via HTTP request. This data is then evaluated and applied to the host.

8.4.254 Technical description

- **Name:** load_remote_host_config
- **Description:** This function establishes a connection with a remote host and fetches configuration details required for the host. It uses 'curl' to send an HTTP GET request to the remote host, and 'eval' to evaluate and apply the configuration data that is received from the host.
- **Globals:** None
- **Arguments:** None
- **Outputs:** If the function cannot fetch or load the host configuration, it outputs a log message "Failed to load host config". If debug is enabled, it outputs the fetched configuration with the log message "Remote config: $conf".
- **Returns:** This function will return `1` if there is any error in fetching the configuration.
- **Example usage:**

  ```
  load_remote_host_config
  ```

8.4.254.1 Code breakdown:

- `local conf` and `local gateway="$(get_provisioning_node)"` are used to declare and initialize local variables.
- `curl -fsSL "http://${gateway}/cgi-bin/boot_manager.sh?cmd=host_get_config"` is used to fetch the configuration information from the remote host.
- `remote_log "Failed to load host config"` logs the error message if there is any issue in getting the configuration.
- `remote_log "Remote config: $conf"` outputs the received configuration for debugging purposes.
- `eval "$conf"` is used to apply the fetched configuration to the host.

8.4.255 Quality and security recommendations

1. Use HTTPS instead of HTTP for the curl request to ensure that the data transmission is secure.
2. Verify the integrity of the received configuration data before evaluating it with eval command. Untrusted data can lead to code execution or security breaches.
3. Add error handling for 'get_provisioning_node' function as the function depends on it.
4. The curl command should have timeouts set to prevent the script from hanging indefinitely in case the remote server does not respond.
5. The function should return different error codes for different types of errors - like failure in getting provisioning node or failure in fetching the configuration. This can help in better troubleshooting.
6. Logging should be improved to detail what type of error occurred - whether it is related to network connectivity, server response, etc. for better issue tracking.

8.4.256 `_log`

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: f7f5d16961fc9339682891b49f2fefad3611a12964b1a3cd095d11a46c108959

8.4.257 Function overview

This Bash function, `_log()`, works as a logging subroutine that outputs a given string to both the console and a remotely configured server. It belongs within a larger script, and it's specific use case relates to logging messages within a ZFS pool creation workflow.

8.4.258 Technical description

- **Name:** `_log`
- **Description:** This function performs logging operations to both a remote log and the system console (if `LOG_ECHO` environment variable is set to 1). It is specifically designed for logging operations regarding the creation of ZFS pools on free disks.
- **Globals:** [`LOG_ECHO`: An environmental variable used to control if the logging output should also be printed to console.]
- **Arguments:** [`$*`: A variable-length list of arguments to be logged. These arguments are usually messages related to the operations performed in the ZFS pool creation process.]
- **Outputs:** Printed statements are sent to the console (if `LOG_ECHO` is set to 1) and the `remote_log` function.
- **Returns:** Nothing directly, but it outputs strings to the console and the remote logging service.

- **Example Usage:** `LOG_ECHO=1 _log "ZFS pool created successfully"`

8.4.259 Quality and security recommendations

1. Always sanitize the inputs before logging to prevent log injection attacks.
2. Ensure that the `remote_log` function correctly handles connection failures and other errors to prevent disruption of the main program.
3. Consider adding timestamp and log level (info, warning, error, etc.) information to the log messages to provide better logging context.
4. Regularly rotate and archive logs to prevent them from occupying too much disk space.
5. Encrypt sensitive data in logs to protect them from being exposed to unauthorized users.
6. Consider implementing rate limiting to prevent DOS attacks via rapid, repeated calls to the `_log` function.

8.4.260 `log`

Contained in `lib/host-scripts.d/common.d/common.sh`

Function signature: 8036c583d5ea28bb7d9be8b86179593acefd5b00fca118ffa37907a177f20aec

8.4.261 Function Overview

The `log` function is designed to output and forward log messages. It prepends the string “HPS” and current time in the “Hour:Minute:Second” format to the log message, then writes the complete string to the standard output. The function also sends the log message to a remote log function called `remote_log`.

8.4.262 Technical Description

- **Name:** `log`
- **Description:** This function takes any number of arguments, appends them after a prepended string which contains “HPS” and the current time. The final string is then echoed (printed to stdout). It also sends every log message to `remote_log` which is assumed to be another logging function.
- **Globals:** No global variables are used or modified by this function.
- **Arguments:** Any set of strings which need to be logged. Example: `$*: log message`.
- **Outputs:** Outputs to STDOUT. Example: `[HPS:14:20:35] your log message`.

- **Returns:** It does not return any value but calls another function called `remote_log`.
- **Example usage:**

```
log "Application started."
```

This will print:

```
[HPS:14:20:35] Application started.
```

and will also send "Application started." as a log message to the remote log.

8.4.263 Quality and Security Recommendations

1. **Secure transfer for remote logging:** The function `remote_log` which is called within this function should ensure secure transmission of log messages, especially if these logs are being sent over a network.
2. **Error Handling:** The `remote_log` function is called without any error handling. If it fails for any reason, there won't be any fallback or even notification given to the user. This needs to be ensured for better quality.
3. **Detailed TimeStamp:** The timestamp is currently pretty simple just HH:MM:SS. For better track of logs, it could be enriched to contain more details like the date, timezone, milliseconds etc.
4. **Input Validation:** This function accepts any string without validation. Although this is generally okay for a logging function, it could be an issue if certain types of strings could cause problems with the `remote_log` function. Consider validating inputs or sanitizing them if necessary.

8.4.264 `make_timestamp`

Contained in `lib/functions.d/system-functions.sh`

Function signature: 7ea1fc9d3621ad0a04879323d75cd0a5f1aa2468bf98b9b3c83ff2b66dfa8e3d

8.4.265 Function Overview

The function `make_timestamp` is a simple Bash function that returns the current date and time in a specific format (Year-Month-Day Hour:Minute:Second UTC). The -u option makes sure that the command uses Coordinated Universal Time (UTC) instead of the local timezone.

8.4.266 Technical Description

- **Name:** `make_timestamp`
- **Description:** This function uses the `date` command with the −u option to get the current date and time in UTC, formatted as: Year-Month-Day Hour:Minute:Second UTC.

- **Globals:** None.
- **Arguments:** None.
- **Outputs:** Prints current date and time formatted as Year-Month-Day Hour:Minute:Second UTC.
- **Returns:** Nothing.
- **Example usage:**

  ```
  $ make_timestamp
  2023-01-01 00:00:00 UTC
  ```

8.4.267 Quality and Security Recommendations

1. This function has no user-input, hence it should be safe from security flaws that could result from unreliable user input.
2. Since the `date` command is a common Unix command, it should be available in most environments. However, in case the environment does not support the `date` command, appropriate error handling could be added.
3. For quality, consider adding checks if UTC timezone is required or if local timezone would suffice. If different timezones are needed, consider making the timezone an optional input to the function.
4. If the function will be used as part of larger scripts, consider returning the value instead of printing to allow better usage of this function.

8.4.268 `normalise_mac`

Contained in `lib/functions.d/network-functions.sh`

Function signature: ac8cefca0a4fe56f9e4ef01e54a13bb17bd1107670d5f1b98c4c04d07fd2425e

8.4.269 Function overview

The `normalise_mac()` function is used to validate and standardize the format of a MAC address. The function performs the following processes: it removes all the common delimiters such as colon (:), dash (-), dot (.), and space ( ). Then, it converts all the characters in the MAC address to lowercase. The last process validates the MAC address. The resulting MAC address should exactly have 12 hexadecimal characters. If the MAC address format is invalid, the function will print an error message and return 1. If the format is valid, it will display the normalized MAC address.

8.4.270 Technical description

```
- name: normalise_mac
- description: A function to validate and standardize MAC
 ↪  addresses. It removes common delimiters, converts to lower
 ↪  case, and validates the address to be exactly 12 hexadecimal
 ↪  characters.
- globals: [ ]
- arguments: [ $1: The MAC address to be normalized and validated ]
- outputs: An error message to stderr if the MAC address is
 ↪  invalid. If valid, the normalized MAC address will be returned.
- returns: The function will return 1 if the MAC is invalid.
- example usage: normalise_mac "00-80-41-ae-fd-7e"
```

8.4.271 Quality and security recommendations

1. Implement higher-level error handling: Rather than only printing to stderr, throw an error that can be caught by other parts of your application.
2. Ensure to sanitize input: Even though the script appears resistant to command injection, proper input sanitization should always be performed when accepting user input.
3. Testing: Add various test cases to confirm the function behaves as expected and is able to handle edge cases and unexpected inputs.
4. Document the function: Add clear and concise documentation to the function explaining its use, inputs, and outputs for better understandability and maintainability of code.
5. Return unique error codes: Return unique error codes for different error cases for easier debugging.

8.4.272 `_osvc_run`

Contained in `lib/host-scripts.d/common.d/opensvc-management.sh`

Function signature: b4d91dc406a4ee6f725ae31a5e96d8ba670b6af35d65b7ad9e0416aecb14c186

8.4.273 Function overview

The `_osvc_run` function is a utility function in Bash that encapsulates the command execution pattern accommodating argument passing, error capture, and logging. This function accepts command arguments, executes them, logs any error codes or output, and returns the error code.

8.4.274 Technical description

8.4.274.1 Name

_osvc_run

8.4.274.2 Description

This script function accepts a description and a command as inputs, and executes the given command. Any resulting output or errors are logged using a custom logger, tagged with a provided description and an exit code. The function then returns the same exit code.

8.4.274.3 Globals

None

8.4.274.4 Arguments

- $1: `desc` - A description tag for the command being run. This gets logged with any output or errors.
- `$@`: The remaining arguments form the command to be executed.

8.4.274.5 Outputs

Logs an information message on execution status including the provided tag, the exit code, and any command output or errors.

8.4.274.6 Returns

The exit status of the command that was run.

8.4.274.7 Example Usage

```
_osvc_run "List running processes" "ps aux"
```

8.4.275 Quality and security recommendations

1. Aside from the facilities provided by Bash, there's no explicit error checking or sanitization. Ensure the commands or arguments that this function wraps are safe to execute.
2. Consider adding checks for the number of parameters provided to the function to avoid undesired behavior.
3. Make sure to properly escape the parameters that are passed to this function to avoid command injection vulnerabilities.
4. Always use the function with trusted inputs, as this function executes commands directly.
5. Ensure that the right permissions are granted at the directory and shell level. The function should not have more permissions than what it needs.
6. Incorporate a controlled logging mechanism to reduce the risk of potential log forgery.

8.4.276 `prepare_custom_repo_for_distro`

Contained in `lib/functions.d/repo-functions.sh`

Function signature: f41fadc94c0ebb53ebb87776324ccfcc73903f665dfe9bfd0a9d3f4c9cbb827c

8.4.277 Function Overview

The function `prepare_custom_repo_for_distro` serves to prepare a custom software repository for a given linux distribution represented by the input `dist_string`. It also takes an array of URLs or local files to download and add to the custom repository. The function segregates URLs and local file paths from the other entries stamped as required packages in an iterative manner. Further, it initiates repository creation, downloads the package from each URL or copies from local files, and builds the YUM repository. It verifies that the required packages are available in the repository and logs an error if a package is missing.

8.4.278 Technical Description

- **Name**: prepare_custom_repo_for_distro
- **Description**: This function prepares a custom software repository for a given distribution by segregating source links (URLs or local file paths) from required package listings, creating repositories, building YUM repository, and ensuring presence of required packages.
- **Globals**: *HPS_PACKAGES_DIR*: The root directory where repositories are created.
- **Arguments**:
  - `$1`: `dist_string`, the string representation of a Linux distribution.
  - `$@`: An array of URLs, local file paths to the required packages or names of the required packages.
- **Outputs**: Logs various informational and error messages during repository preparation. Copies or downloads package files to the repository.
- **Returns**: The function uses return values varying from 0-6 to indicate success or various types of failures (such as, failure to create repository directory, download or copy a package, etc.).
- **Example usage**: `prepare_custom_repo_for_distro "ubuntu" "https://example.com/package1.deb" "package2" "/path/to/package3.deb"`

8.4.279 Quality and Security Recommendations

1. **Error Handling**: At present, the function returns error messages in various places but a more rigorous error handling system should be implemented for each step.
2. **Input Validation**: Currently, the repository name and the package sources are not validated. They should be confirmed to avoid unpredictable behavior.

3. **User Permissions**: Permissions required to create directories or copy files need careful considerations.
4. **Logging & Auditing**: It would be good to maintain consistent logging throughout the function to account for all actions taken.
5. **Dependency Handling**: The function should manage potential dependencies of the required packages.
6. **Use Secure Communication**: If possible, use secure protocols (like HTTPS) for downloading packages from URLs.

8.4.280 `reload_supervisor_config`

Contained in `lib/functions.d/configure-supervisor.sh`

Function signature: 1ad96e9487bac5b94d7eca312662f18392454507c7cddfac5de26d924e922722

8.4.281 Function overview

The `reload_supervisor_config` function is designed to allow for the reloading of Supervisor configuration in a Bash environment. This function takes no arguments. It operates by re-reading the configuration contained in a directory defined by `HPS_SERVICE_CONFIG_DIR` and updating the system Supervisor accordingly. Should there be changes in the Supervisor configuration, this function will allow Supervisor to adopt the new configurations in real-time without the necessity of a full system restart.

8.4.282 Technical description

- Name: `reload_supervisor_config`
- Description: A bash function to reload the configuration for Supervisor by re-reading the configuration file from a specified location and then updating Supervisor. It allows for changes in configuration to take effect in real-time without requiring a full reboot or restart.
- Globals: [`HPS_SERVICE_CONFIG_DIR`: The directory containing the configuration file for Supervisor.]
- Arguments: The function does not accept any arguments.
- Outputs: Does not output any value. However, it re-reads and updates the Supervisor configuration file.
- Returns: N/A
- Example Usage:

```
HPS_SERVICE_CONFIG_DIR="/path/to/config_dir"
reload_supervisor_config
```

8.4.283 Quality and Security recommendations

1. Implement error handling: To improve the function's robustness, introduce error handling to deal with potential issues like non-existent configuration directories or issues with supervisorctl.
2. Ensure correct permissions: Only trusted users should have access to run this function and ensure that the `HPS_SERVICE_CONFIG_DIR` directory has the correct permissions to prevent an unauthorized user from tampering with the configurations.
3. Validate the configuration file: While loading the configuration file, ensure that the file is not corrupt and is in the correct format to prevent potential issues during its use.
4. Implement logging: To effectively troubleshoot issues that may arise when reloading the configuration, make sure to log both successful operations and errors. This way, any errors that occur while reloading the configuration can be easily traced and resolved.

8.4.284 `remote_cluster_variable`

Contained in `lib/host-scripts.d/common.d/common.sh`

Function signature: 80d48ac6c7e7f4246491425bf9e89a589b854a03598f7fff94cfddfd4f9b7a55

8.4.285 Function Overview

The function `remote_cluster_variable` is essentially used for manipulating the cluster variable in a remote Bash environment. It enables both the reading (GET) and writing (SET) of a cluster variable over HTTP, through a POST or GET request. The function communicates with the server via curl and utilizes a gateway system as the intermediary node for communicating with the remote cluster. The variable's value is URL-encoded before it's sent, ensuring its safe passage across the network.

8.4.286 Technical Description

- **Name:** `remote_cluster_variable`
- **Description:** The function manages a cluster variable in a remote bash environment. If two arguments are provided, it sets (POSTs) the specified value of the variable. If only one argument is provided, it gets (GETs) the value of the variable.
- **Globals:** None
- **Arguments:** `$1: name` (The name of the variable. If it's not provided, the function raises an error and terminates), `$2: value` (The value to set to the variable. This argument is optional. If it's provided, function sets (POSTs) this value to the variable. If it's not provided, function gets (GETs) value of the variable)
- **Outputs:** The function outputs the response from the POST or GET request.

- **Returns:** Returns 1 if there's an error in `get_provisioning_node` function. Otherwise, nothing is returned explicitly.
- **Example Usage:** `bash remote_cluster_variable username john remote_cluster_variable username`

8.4.287 Quality and Security Recommendations

1. Always URL encode all inputs to prevent potential security threats such as injection attacks.
2. Error checking for all variable assignments and function calls should be implemented for greater resilience.
3. Consider implementing retry logic for the curl command, as network requests are oftentimes flaky and may need to be retried.
4. Implement logging to catch and debug potential errors during runtime.
5. Make sure that the gateway system, being the relay of data, is secure against possible threats.
6. Validation for the user-inputted name and value, such as type or format restrictions, could be considered.
7. Always use HTTPS for such requests to ensure the privacy and data integrity of your communications.

8.4.288 `remote_function_lib`

Contained in `lib/functions.d/kickstart-functions.sh`

Function signature: b6d9cd335e4f61b186f614aa07279b2f8bdd77298bf573a231ffd21869c18118

8.4.289 Function Overview

`remote_function_lib` can be described as a "meta" or higher-order function that essentially creates and exports a library of Bash functions. This code block is designed to synthesize a set of functions that can be injected into the pre and post sections of a remote script, affording a greater degree of modularity and reusability in a Bash-based program. Given the comment, this function is apparently a single point of reference for these functions, but there is a suggestion to move it to its own separate file for more organize project structure.

8.4.290 Technical Description

- **Name:** `remote_function_lib`
- **Description:** This function generates functions that are intended to be injected into pre and post sections of a remote script.
- **Globals:** None
- **Arguments:** None.

- **Outputs:** It outputs the injection functions in a here-document format.
- **Returns:** No values are returned directly from the function.
- **Example usage:** `remote_function_lib`

8.4.291 Quality and Security Recommendations

1. The commenter recommends considering moving this library into its own separate file. This is indeed a good practice, especially in larger codebases, because it can help to modularize the code and improve the overall maintainability of the codebase.
2. The function does not currently accept any arguments. Depending on how flexible the library of functions needs to be, it may be beneficial to add parameters that can customize its creation.
3. While here-documents are a valid and efficient way to create multi-line outputs in Bash, injecting functions in this way could potentially open up security concerns, such as code injection vulnerabilities, depending on how the output is used. It is recommended to carefully sanitize and validate any input that could end up being executed as code.
4. As a general security practice, it may be helpful to add error checking within the function, or even return meaningful error messages if something in function library creation process goes wrong. This could prove to be beneficial in troubleshooting potential issues.

8.4.292 `remote_host_variable`

Contained in `lib/host-scripts.d/common.d/common.sh`

Function signature: d808c9948262b9c10e2d29dbf91fff37d9e5c1ad4ce5c78af73813e01521b322

8.4.293 Function overview

This Bash function, `remote_host_variable`, is designed to interact with a remote host's boot manager. The function sets a local variable for the name, and optionally for the value. The gateway is set to the function call of 'get_provisioning_node'. If an error is found, the function will immediately return with a status of 1.

If a second argument is provided, it is encoded and added to the URL along with the encoded name. A POST request is made to this URL, which presumably interacts with the remote host, potentially setting a variable or performing some other change.

8.4.294 Technical description

- Name: `remote_host_variable`
- Description: This function performs an HTTP POST request to a remote boot manager, potentially setting a variable.

- Globals: None
- Arguments: [`$1`: This is the name of the variable to be set or changed, a mandatory input. `$2`: This is the optional value of the variable.]
- Outputs: Outputs are dependent on the return of the `curl` call.
- Returns: If the `get_provisioning_node` function call encounters an error, the function returns 1. Otherwise, return value is determined by the `curl` request.
- Example usage: `remote_host_variable "NAME" "VALUE"`

8.4.295 Quality and security recommendations

1. This function relies on a successful return from 'get_provisioning_node', but does not validate its output in any way. You should add a check to ensure that 'get_provisioning_node' returns a valid result.
2. There is a potential risk of URL injection with the `$2` argument. Ensure the validation and sanitization of the `$2` and `$1` parameter, especially if using this function in web applications or other programs where user input might be used as these arguments.
3. Rather than forming your URL by concatenation, consider using an URL-building library function for a more robust and error-resistant approach.
4. Always ensure that the correct headers are used in your curl requests to ensure authenticity and data integrity when communicating with the remote server.
5. The URL base-string used in the function is hardcoded into the code itself, consider moving it out to a configuration file or argument to make the script more versatile.
6. Consider using HTTPS for secure communication instead of HTTP.

8.4.296 `remote_log`

Contained in `lib/functions.d/kickstart-functions.sh`

Function signature: 1f586056ba1b573eed69d32639c3940c1c742ba73af4aae917a1d65d36c5367c

8.4.297 Function Overview

The bash function named `remote_log` is used to encode a message into URL format and then send it as a log message to a remote server using a POST request.

8.4.298 Technical Description

- **Name:** `remote_log`
- **Description:** This function accepts a string (log message) as an input argument, encodes it into URL format, and then sends it as a POST request to a remote server using `curl`.
- **Globals:**
  - `HOST_GATEWAY`: the gateway of the host machine.

    - `macid`: the machine identifier.
- **Arguments:**
    - `$1`: The log message to be URL-encoded and sent to the remote server.
- **Outputs:** Sends a URL-encoded log message using a POST request to a URL specified using `HOST_GATEWAY`, with additional parameters including `macid`.
- **Returns:** None. It only initiates a `curl` POST request but doesn't handle the response.
- **Example Usage:**

```
remote_log "This is a log message"
```

8.4.299 Quality and Security Recommendations

1. URL-encoding should be improved to handle special characters in a more comprehensive and foolproof way.
2. The URL target for the `curl` POST request should be validated to ensure it's well-formed and secure to connect to.
3. Error handling should be added to ensure that the function behaves predictably when things go wrong. For example, handling the case when the message is empty or the target URL refuses connection.
4. The use of global variables could be removed or limited for better code encapsulation and reusability. In scenarios where globals are necessary, they should be validated before use.
5. Secure protocols such as HTTPS should be used instead of HTTP to ensure better security in data transmission.

8.4.300 `rtslib_add_lun`

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: b7dcfb5ba5fb6cf52d4e8f8990eea725726e0de361be466ec9a246c28e59ba62

8.4.301 Function Overview

The function `rtslib_add_lun()` is a bash script function that interacts with a Python script. It adds a Logical Unit (LUN) to a specific target in the `rtslib-fb` Python module. A LUN is a logical reference to a portion of a storage subsystem. This function is used in the configuration of iSCSI (Internet Small Computer System Interface) targets, which allows the sharing of storage resources over a network.

8.4.302 Technical Description

- **Name:** `rtslib_add_lun`
- **Description:** This function adds a Logical Unit (LUN) to an iSCSI target in the `rtslib-fb` Python module using a Python script. The Python script checks to

see if the target exists by comparing `iqn`s. If the target is discovered, the LUN is appended to the list of that target's LUNs.
- **Globals:** [] There are no global variables.
- **Arguments:**
  - `$1: remote_host`, it is the name of the remote host.
  - `$2: zvol_path`, refers to the path to the zvol (ZFS volume).
- **Outputs:** If the target is not found, the script will print "⊠ Target not found".
- **Returns:** The function doesn't directly return anything since it is used to manipulate state in the Python `rtslib-fb` module rather than produce output within the bash script. However, a side effect is that the Python script will cause an exit with status 1 when the target is not found.
- **Example Usage:** `rtslib_add_lun "remotehost" "/path/to/zvol"`

8.4.303 Quality and Security Recommendations

1. The function could benefit from more sophisticated error handling. For instance, adding more explicit checks for whether the command-line arguments `$1` (`remote_host`) and `$2` (`zvol_path`) were provided.
2. This function relies heavily on Python script, it is recommended to preserve dependencies and ensure that the used python modules are kept up-to-date for the overall system's security.
3. It would be prudent to add validation to ensure that the `zvol_path` provided as an argument leads to a real and accessible path.
4. For better security, the function should handle cases where there are multiple iSCSI targets with the same iqn. This could involve updating the process to accept an additional parameter to uniquely identify targets.

8.4.304 `rtslib_check_available`

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: 600db02b1741924e983f7cfb8291447bd0b965ffffa533f6f2b35a4b9ae08b44

8.4.305 Function overview

The `rtslib_check_available` function checks if the python3 rtslib_fb module is available. If the module is not available, the function will output a text message, indicating the absence of the module. If the module is found, nothing happens and the function returns 0, which signifies success. The function is executed in the current shell, meaning any changes made happen within the current shell.

8.4.306 Technical description

Name: `rtslib_check_available`

Description: This function checks whether the module `python3-rtslib_fb` is accessible. If the module is unavailable, it outputs a message notifying the user and returns 1. If the module is available, the function returns 0 and no message is output.

Globals: None

Arguments: None

Outputs: If the module `python3-rtslib_fb` is not available, it outputs "⊠ python3-rtslib_fb is not available".

Returns: - `1` if the `python3-rtslib_fb` module is not available. - `0` if the module is available.

Example usage:

```
rtslib_check_available

# Expect output if rtslib_fb isn't present:
# ⊠ python3-rtslib_fb is not available
```

8.4.307 Quality and security recommendations

1. Where possible, ensure that all external modules used in the function are up-to-date. This is to mitigate possible security issues that come with using outdated modules.
2. Error handling helps maintain the quality of the code. Check for potential errors and handle them properly. In this case, the function handles the error that arises from the `python3-rtslib_fb` module not being available.
3. Using a standard success/failure status like this function does (returning `0` for success and 1 for failure) is also a good practice. This makes the function's usage easier for other developers or scripts checking for success.
4. Lastly, do thorough testing on various environments because a module available on one machine or in one environment may not be available on another.

8.4.308 `rtslib_create_target`

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: 05a02a7bcad8e26b1c77c0b01e3325bd5fd95fd6fc531cb13f7285b33082a1df

8.4.309 Function Overview

The `rtslib_create_target()` function is primarily used to create an iSCSI target. The function establishes the iSCSI target by invoking Python 3 from within a Bash shell, utilizing the `rtslib_fb` Python library with preset attributes and saving them to a file. The iSCSI target is given a name based on the remote_host variable and the current date.

8.4.310 Technical Description

- **Name:** `rtslib_create_target()`
- **Description:** This function creates an iSCSI target by using Python 3 and the `rtslib_fb` Python library. The function takes in a remote host as an argument and creates an iSCSI target name determined by the current date and the provided remote host.
- **Globals:** None
- **Arguments:**
  - `$1`: The name of the remote host
- **Outputs:** Either creates an iSCSI target or returns a failure message.
- **Returns:** Nothing
- **Example usage:** `rtslib_create_target 192.168.1.15` will create an iSCSI target related to the provided IP address.

8.4.311 Quality and Security Recommendations

1. Proper Validation: The function should have checks put in place to ensure that the input, in this case, the remote host, is correctly validated and sanitized. This can help prevent any sort of injection or misconfiguration.
2. Error Handling: Increase robustness of the function by putting more specific error handling with clearly defined error messages.
3. Secure attribute setting: Be mindful of the iSCSI target attributes being set here. For example, this function disables authentication (`"authentication", False`), which may not be desirable in a secure setup. Check and review these attribute settings to better suit them to your environment.
4. Use of globals: This function does not use any global variables which is a good practice in terms of code clarity and avoiding unexpected side effects. However, if any are to be used in the future, they should be carefully managed.
5. Logging: Proper logging techniques need to be followed so that any issue can be debugged effectively. In the absence of effective logging, it will be difficult to trace the issue in the case of any failure.

8.4.312 `rtslib_delete_target`

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: 318b4dea0022039da24b23ab7ac595a5bfd08e12b0ec24fa41fd9d6813b14435

8.4.313 Function Overview

The Bash function `rtslib_delete_target()` is designed to delete a specific target on a remote host with the help of an iSCN (iSCSI Qualified Name). This function takes a remote host as an argument and generates an iSCN with the remote host embedded in it. Later, it uses python3 to implement the SCSI protocols with the help of the rtslib library for Python, deletes the required target, and saves the updated configuration to a file. If the target doesn't exist, it will print a message saying the target is not found.

8.4.314 Technical Description

- **Name**: rtslib_delete_target
- **Description**: This function deletes an ISCSI target provided by the remote host. It will print a confirmation message if the deletion is successful or a fail message in case the target is not found.
- **Globals**:
  - [VAR: iqn]: It describes the iSCSI qualified name dynamically generated using the current year and month, appending the remote host name provided. It helps as the unique identifier for identifying the iSCSI targets.
- **Arguments**:
  -
- **Outputs**: Prints a confirmation message indicating whether the deletion action was successful. It either shows a checkmark with 'Target deleted' or a cross sign with 'Target not found'.
- **Returns**: Nothing.
- **Example usage**: `rtslib_delete_target my_remote_host`

8.4.315 Quality and Security Recommendations

1. Ensure that the remote host's input is correctly sanitized to avoid code injection or other kinds of security issues.
2. The function should have an error handling mechanism for scenarios where the Python code fails to execute.
3. Although the script currently handles targets not found gracefully, it could be further improved to provide more specific error messages.
4. It is essential to have permission checks so that unauthorized users cannot delete targets.
5. Ensure that date command used to generate iqn is correctly referenced in terms of the timezone, as it may create discrepancies due to timezone differences.

8.4.316 `rtslib_list_luns_for_target`

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: 2e1e32139249bea08ce91bc7a5e64b08a087499b34dbb1b776cbd4e7d369d1e0

8.4.317 Function overview

The `rtslib_list_luns_for_target` is a Bash function designed to list the Logical Unit Numbers (LUNs) for a specific iSCSI target. It accepts two arguments, generates an iSCSI Qualified Name (IQN) for the target, and then uses a Python script to iterate through the LUNs. Information on each LUN is then outputted, or an error message is displayed.

8.4.318 Technical description

- **Name**: `rtslib_list_luns_for_target`
- **Description**: This bash function lists the LUNs for a particular iSCSI target. It achieves this by launching an embedded Python script which uses the *rtslib_fb* library to enumerate over the LUNs for a target.
- **Globals**: None
- **Arguments**:
  - `$1`: The remote host. Used to construct the iqn.
- **Outputs**: Depending on the results of its operations, the function either outputs each of the LUNs for the specified iSCSI target or an error message when the target is not found.
- **Returns**: Nothing
- **Example Usage**: `rtslib_list_luns_for_target "localhost"`

8.4.319 Quality and security recommendations

1. Ensure that the remote host argument is appropriately sanitized to prevent potential command injection attacks.
2. Consider handling or logging the captured Python exceptions for debugging and traceability.
3. Look into dealing with situations where the Python environment or the required libraries are not available.
4. Assess whether the function behaves as expected if called with an invalid or unreachable host.
5. Consider how the function will respond if `date` cannot provide the expected output.
6. Check that the user running this script has the necessary privileges for the script's operations.
7. Enforce strict mode in the bash script to minimize potential errors due to uninitialized variables or unhandled errors.

8.4.320 `rtslib_list_targets`

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: 6e4bf5d3b011e59cd3a4346f13ded587844ffead7fe935f5f5e08c258f5d737e

8.4.321 Function Overview

The `rtslib_list_targets` function is a Bash function that is used to list all the target World Wide Names (WWNs) in the fabric module "iscsi". The function uses Python3 code embedded in a Bash function to interact with the RTS (Runtime Simple) library.

8.4.322 Technical Description

- **Name:** `rtslib_list_targets`
- **Description:** This function is designed to list all the target WWNs in the fabric module named "iscsi". Internally, it uses a small Python3 script that utilizes the `RTSRoot` class from the `rtslib_fb` (RTS Library Full-Blown) module. This class provides an interface to the root of the RTS configuration model. The function iterates through each target of the RTS root, checks if its fabric module's name is "iscsi", and if so, prints its WWN.
- **Globals:** None.
- **Arguments:** None.
- **Outputs:** Prints the WWNs of all targets in the "iscsi" fabric module to stdout.
- **Returns:** None. Outputs are sent directly to stdout.
- **Example Usage:** To use this function, simply source it in your bash script and call it without any arguments. The function will print the output to stdout, so you may wish to capture that in a variable or pipe it to another command for further processing: `sh source rtslib_list_targets.sh rtslib_list_targets`

8.4.323 Quality and Security Recommendations

Here are some suggestions to improve the quality and security of the function:

1. **Input Validation:** Even though this function does not accept any arguments, checking for unexpected inputs is always a good habit. Always make sure that the environment in which you are running your script is safe and secure.
2. **Error Handling:** Add error handling to the function to deal with potential issues that may arise, such as failure to import the required Python module or issues accessing the RTS configuration root.
3. **Documentation:** Enhance the code readability by adding more comments within the function explaining what each line or block of code is meant to do.
4. **Security:** This function operates with the RTS configuration model and may have effects on the network configuration. Make sure it is only operative under necessary permissions and not susceptible to unauthorized execution.

8.4.324 `rtslib_save_config`

Contained in `lib/host-scripts.d/common.d/iscsi-management.sh`

Function signature: ac265647656df50229187ef098655cc149e59be37bf19e8244f9356fddcace3e

8.4.325 Function Overview

The `rtslib_save_config` function creates a Python3 child process and runs an inline (heredoc) Python script. This script imports the `RTSRoot` class from the `rtslib_fb` module and calls the `save_to_file()` method on an instance of that class. Therefore, using this function writes the current state of the runtime configuration of the target (presumably a storage target/subsystem) to the configuration file.

8.4.326 Technical Description

- **Name**: `rtslib_save_config`
- **Description**: The function calls a Python3 subprocess, importing the `RTSRoot` class from the `rtslib_fb` module within the Python environment. It then invokes the `save_to_file` function of a `RTSRoot` class instance, thus saving the current state of the target's runtime settings into a file.
- **Globals**: None.
- **Arguments**: None.
- **Outputs**: Writes the target's current runtime settings into a file.
- **Returns**: None. If the Python subprocess encounters an error, it will be written to stderr.
- **Example Usage**:

  ```
  rtslib_save_config()
  ```

8.4.327 Quality and Security Recommendations

1. Ensure proper exception handling is done. While using the Python code, if there is any exception caused due to environment issues or some runtime exceptions, this script would fail. So, appropriate error handling will make the function more robust.
2. Validate Python version: Although the script specifically asks for Python3, there may be differences between minor versions. It's always a good idea to make sure the right version of Python is available.
3. Document the "rtslib_fb" module and its usage: If the person using this function isn't familiar with the "rtslib_fb" module being imported, they could incorrectly use or modify this function. A brief explanation of the module and its usage would be helpful.
4. Use a more secure way to call Python scripts: While Bash is a flexible language, it also can have security implications when running scripts. Always consider the security of your Python scripts when using them in a Bash environment.

8.4.328 `script_render_template`

Contained in `lib/functions.d/kickstart-functions.sh`

Function signature: 064ddcb49f3688c1b0ede8ce884eae36c9ac96f5117338bdce1f62d5c6960a67

8.4.329 Function Overview

The `script_render_template()` function is designed to remap all variable placeholders (`@...@`) with their corresponding values (`${...}`). The function cycles through all the variables using `compgen -v`, assigns the value of the variable to a local variable, and then adds it to an array of values for awk. The awk utility then processes the array of values, replacing each placeholder in the original string with the corresponding variable value.

8.4.330 Technical Description

- **Name**: `script_render_template`
- **Description**: This function is used to remap variable placeholders with their actual value. It does this using the awk utility and a for loop iterating over all variables in the scope.
- **Globals**: None
- **Arguments**: The function does not require any arguments. It acts on all variables in its scope.
- **Outputs**: The function outputs a string with all `@var@` placeholders replaced with their corresponding `${...}` values.
- **Returns**: The function does not have a return value.
- **Example Usage**: Assume that the following variables are already defined in the context:

  ```
  script_name='MyScript'
  script_version='1.0'
  ```

  If we call `script_render_template` in a context where a template string like `'This is @script_name@ version @script_version@'` is present, the function will output the string `'This is MyScript version 1.0'`.

8.4.331 Quality and Security Recommendations

1. Always make sure the substitution values (`${...}`) are securely obtained and sanitized to prevent command injection attacks.
2. Consider validating the variable names that `compgen -v` produces to ensure they adhere to expected patterns and rules.

3. Beware of potential performance issues if the function is used in a context with a large number of variables.
4. Handle errors and exceptions gracefully. For instance, what should happen if a placeholder variable does not exist?
5. Ensure that the function fits well within your specific use case, as its current implementation is very general and may not be suitable for more specific tasks.

8.4.332 `_supervisor_append_once`

Contained in `lib/functions.d/configure-supervisor.sh`

Function signature: 5f79fe5e7a4a7d3eb591b9e371a140f9bd1de60abf5e87d0e47082bfff056191

8.4.333 Function Overview

The function `_supervisor_append_once` is designed to modify the settings for Supervisor, a control system for UNIX-based servers. It takes two arguments—`stanza` and `block`. The function checks the supervisor configuration file, which is determined by the environment variable `${SUPERVISORD_CONF}`. If the stated `stanza` is not present in the configuration file, it then appends the corresponding `block` at the end of the file. This function is used multiple times subsequently in the script to ensure Supervisor is configured to securely and efficiently manage various service programs.

8.4.334 Technical Description

- **Name**: `_supervisor_append_once`
- **Description**: This function appends configuration blocks to the Supervisor configuration file for a specified service program, but only if that block does not already exist. It is used in managing UNIX based servers.
- **Globals**: [`${SUPERVISORD_CONF}`: Path to the Supervisor configuration file]
- **Arguments**:
  - `$1`: `stanza`— the name of the service program, e.g. `program:nginx`
  - `$2`: `block`— the complete configuration string to be appended.
- **Outputs**: Appends a configuration block to the Supervisor configuration file for a specific service program.
- **Returns**: Nothing.
- **Example Usage**: `_supervisor_append_once "program:dnsmasq" "$(cat <<EOF`
  - This usage of the function appends dnsmasq related configurations to the Supervisor configuration file.

8.4.335 Quality and Security Recommendations

1. Enhance error handling to prevent the potential mishandling of non-existent files.

2. Introduce a mechanism to backup the original configuration file before making changes to it.
3. Expand on arguments validation by checking for empty or null arguments.
4. Since shell scripts are susceptible to injection attacks, consider potential sanitization steps for the `stanza` and `block` variables.
5. Avoid hard coding paths and consider converting them into arguments or environment variables to improve scalability.

8.4.336 `ui_clear_screen`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: 0493ca3490c6e95475fb08d2f387298f827857616ec67718e0dd7bc3268a31d2

8.4.337 Function Overview

The `ui_clear_screen()` function is a simple Bash utility function used to clear the terminal screen. If the `clear` command is available and successful, it will be used. Otherwise, it will fall back to outputting an escape sequence that should also perform the same function, this is especially handy in various system environments where the `clear` command may not be available.

8.4.338 Technical Description

- Name: `ui_clear_screen`
- Description: This function is responsible for clearing the terminal screen. It first tries to use the `clear` command and falls back to an escape sequence (`\033c`) if `clear` is not successful.
- Globals: Not applicable as this function does not use or modify any global variables.
- Arguments: Not applicable as this function does not take any arguments.
- Outputs: A cleared terminal screen.
- Returns: If the `clear` command is successful, this function will return the exit status of the `clear` command which is expected to be 0 indicating success. If the `clear` command fails (non-zero exit status), 0 will be returned after printing the escape sequence.
- Example Usage: `bash   ui_clear_screen()`

8.4.339 Quality and Security Recommendations

1. Always ensure that functions correctly handle unexpected or erroneous input. For `ui_clear_screen()`, that’s not necessary as it doesn’t take any arguments.

2. The function doesn't provide or log any error messages which might be useful in some scenarios to debug issues related to the terminal or environment. You could consider adding error logging.
3. Evaluate the fallback method of using an escape sequence for screen clearing, it might not work or could lead to unexpected results in certain terminal environments. A more robust way of handling the unlikely event of `clear` failing could be devised.
4. Make sure to check the return values of commands you are running within your functions, incorporate error checking mechanisms wherever possible.
5. Consider outputting a warning or notice to the user when the fallback method is employed letting them know `clear` command wasn't successful.
6. This function should be safe from code injection as it does not process external input or environmental variables.

8.4.340 `__ui_log`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: d3d49681e2b691a5ff908ab2cfc80feb43c6f2c7f2aa6c60521377957fc9244b

8.4.341 Function Overview

The function `__ui_log` in Bash is a utility for logging or displaying messages with a distinctive prepend label. It can be utilized to show system-level or user interface related informational, warning, or error messages. The function outputs the passed messages to the standard error stream.

8.4.342 Technical Description

- **Name:** `__ui_log`
- **Description:** This is a logging function used to display messages with a distinctive label '[UI]'. It's mainly intended for outputting system or user interface messages. The function employs the `echo` command to output the messages, which are passed in as arguments.
- **Globals:** None
- **Arguments:** `$*` - A list of arguments to be logged or displayed. They're concatenated into a single string by the `echo` command.
- **Outputs:** The function redirects its output to the standard error output stream (>&2). Hence, any message passed to this function would appear in the stderr stream, with '[UI]' as a prefix.
- **Returns:** None. The function doesn't explicitly return a value.
- **Example Usage:**

```
# Example to log a message
__ui_log "This is a UI log message"
```

8.4.343 Quality and Security Recommendations

1. For added clarity and readability, it could be beneficial to add comments within the function to explain what each component does.
2. The function might be enhanced with the addition of explicit return values, aiding in error handling and flow control in scripts using this function.
3. To prevent command injection attacks, ensure that all variable data is properly escaped before it is included in the log message.
4. Implement a mechanism to control the log level. Thereby, control what kind of messages (debug, info, warning, error) should be directed to the output.
5. Always make sure that no sensitive data is logged in order to maintain information security and privacy.
6. Consider directing the logs to a specific file or a log management system, for easier troubleshooting and better performance in large systems.

8.4.344 `ui_menu_select`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: 228d39d076d4308652684c61dd46d71781022466ed70155a37a899acdc69da71

8.4.345 Function Overview

The `ui_menu_select()` function in Bash is used to create a user interface menu that accepts input from the user and presents a list of selectable options. On selection of a valid option, the function outputs the chosen value and gracefully exits. In case of an invalid selection, it prompts the user for a valid selection until a valid option is selected.

8.4.346 Technical Description

- **Name:** `ui_menu_select`
- **Description:** A bash function that prints a list of options (menu) to the console, takes user input, and validates the input. If the input is valid, it prints the selected option and returns. If the input is invalid, it asks for a new input from the user.
- **Globals:** None
- **Arguments:**
  - `$1`: The prompt string for the UI menu.
  - `$@`: An array containing the selectable options for the UI menu.
- **Outputs:** Prints to stdout either the prompt and options for the UI menu, the selected option on valid input, or an error message on invalid input.
- **Returns:** Returns 0 on success, no explicit return on failure.

- **Example Usage:**

```
options=("option1" "option2" "option3")
ui_menu_select "Please select an option:" "${options[@]}"
```

8.4.347 Quality and Security Recommendations

1. Add checks to validate the input arguments—especially check if the supplied options are not empty.
2. Handle signal interrupts for better robustness.
3. It’s generally a good practice to avoid the use of `echo -n` since its behavior might be different across different systems.
4. Sanitize error messages to avoid misleading information or potential injection vulnerabilities.
5. Consider adding a timeout for user input to prevent potential denial of service if the script is being run as a server-side script.
6. Use unset to destroy variables that store sensitive data after their use to prevent unintentional exposure or leakage of such data.
7. Exit with an error code on failure instead of just printing an error message to indicate error status to the calling script or function.

8.4.348 `ui_pause`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: 1636563cd29eae7fb011743bbb84e1bd4b67567168166cfc3cd0cf46472383ec

8.4.349 Function overview

The Bash function `ui_pause()` is designed to introduce a pause in the script execution, requiring a user intervention to continue. This behavior is accomplished by using the `read` command with the `-rp` option, which reads a line from standard input and prompts with a message until input is received.

8.4.350 Technical description

- **Name:** `ui_pause`
- **Description:** This function uses the `read` command to pause the execution of a script and display a prompt to the user, requesting them to press the [Enter] key to continue.
- **Globals:** None.
- **Arguments:** None.
- **Outputs:** Outputs a message to the user prompting them to press the [Enter] key.
- **Returns:** Does not return a value.
- **Example Usage:**

```
ui_pause
# The script will pause here until the user presses [Enter].
```

8.4.351 Quality and security recommendations

1. Lack of user input validation: In this function, any key strike will be interpreted as a signal to continue execution. It would be advisable to include some level of user input validation to ensure that only the specific [Enter] key press is given the ability to continue.
2. Error handling: Unexpected errors or exceptions during the function execution are not accounted for, introducing the potential for unexpected behavior and script crashes. A recommended improvement would be to include error handling mechanisms within the function code.
3. Usability: The current function prints a static message which may be unclear to some users or not applicable in all scenarios. Allowing customization of the pause message could improve usability.
4. Return value: Even though this function does not need to return a specific value, for consistency with other Bash functions, it may be beneficial to return a success status (0) after successful execution.
5. Security: As this function does not make use of any user-provided data or values, there are no apparent security risks. However, it's always a good practice to keep security in mind and follow safe coding practices throughout.

8.4.352 `ui_print_header`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: 57e2fede290b133fd0e31c859a63c119ad15a0eea0326adcc61d2c65983dfecc

8.4.353 Function Overview

The `ui_print_header()` function in Bash is a utility function built for printing headers in terminal-based user interfaces. The function itself is quite straightforward - it accepts a string as an argument which is then displayed as a title within a border of equals signs (=) before and after the title.

8.4.354 Technical Description

Name: `ui_print_header()`
Description: This function prints a passed title surrounded by a set of equals signs (=) on the lines before and after the title. This creates a clear and visually distinct header within a terminal or console.
Globals: None
Arguments:

- `$1: title` - This is a string placeholder that the function expects. This value is what the function will print out as the header text.

Outputs: This function prints to stdout. The printout consists of an empty line, then a line of equals signs, followed by the title, another line of equals signs, and finally, another empty line.
Returns: Returns nothing.
Example Usage: `ui_print_header "Welcome to My Program"` - Will print:

```
===================================
   Welcome to My Program
===================================
```

8.4.355 Quality and Security Recommendations

1. Be aware that there are no sanity checks on the supplied argument. The function will print whatever is supplied as an argument, making it susceptible to potentially handling unexpected or rogue inputs. Therefore, consider validating or sanitizing the input on a higher level of function call.
2. This function does not check the length of input strings. If an excessively long string is supplied as an argument it can lead to inconsistent formatting and potentially unreadable headers.
3. The function doesn't use the locale settings to determine the orientation of the symbols. This may cause issues when it is used in locales that use right-to-left writing systems.
4. As the function doesn't return anything it would not be suitable for scenarios where error handling or feedback would be required based on the output of the function.

8.4.356 `ui_prompt_text`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: 698fb0f6a52fc5fb7d346cb2755ab85d4e44cef66d1ac20f8f40870457b2970a

8.4.357 Function Overview

The `ui_prompt_text` function in Bash is designed for presenting an interactive prompt to users. This function receives two arguments - a prompt message and a default value. The function first displays the prompt, afterwards if the default value is nonempty, it is also displayed in brackets. A colon and a space character are appended to prepare the text for an expected user input. The user's input is read and stored in a variable, `result`. In case of no user input, the default value is returned, otherwise the user's input is returned.

8.4.358 Technical Description

Feature	Details
Name	ui_prompt_text
Description	A Bash function to prompt users for input with the option of a default response.
Globals	None
Arguments	$1 (prompt): The message prompt to present to the user. $2 (default): The default value that will be used in case of absence of user input.
Outputs	Prompts the user with a message and optional default value.
Returns	The user's input if provided, otherwise the default value.
Example Usage	`ui_prompt_text "Please enter your name" "John Doe"`

8.4.359 Quality and Security Recommendations

1. Always use `read -r` to prevent interpreting backslashes as escape characters.
2. Beware of potential security risks of command injection if the result is used in further commands without sanitization.
3. You should always quote your variable substitutions like so: `"$var"`. This is to prevent issues with multi-word strings.
4. Remember to initialize local Bash variables. This can help avoid problems if there's a global variable with the same name.
5. Provide clear and user-friendly prompts to facilitate the operation for end users.
6. Default values should be carefully chosen to prevent problems in case of user misuse or misunderstanding.
7. When handling sensitive data, ensure that input is hidden or obscured to protect it from unauthorized access or exposure.

8.4.360 `ui_prompt_yesno`

Contained in `lib/functions.d/cli-ui.sh`

Function signature: 5b56e1af7169ad6721f24f1f595cee61d779f0b22963936d53073e284b177e9c

8.4.361 Function Overview

The `ui_prompt_yesno` function in Bash is a user interface function that prompts the user with a yes/no question. It loops until the user provides a valid response. The first argument is the prompt text and the second optional argument sets the default answer.

8.4.362 Technical Description

- **Name:** `ui_prompt_yesno`
- **Description:** This function prompts the user with a question and loops till it gets a valid "yes" or "no" answer from the user. It ensures the users interaction in a script where a binary input is necessary for further execution.
- **Globals:** No global variables are used.
- **Arguments:**
  - `$1`: This is the prompt text that is to be displayed to the user.
  - `$2`: This optional argument specifies the default answer.
- **Outputs:** Outputs the prompt question with an optional default value and user response.
- **Returns:**
  - Returns 0 if the response is "yes".
  - Returns 1 if the response is "no".
- **Example Usage:** If we need user's confirmation for proceeding further, we can use the function as follows:

```
ui_prompt_yesno "Do you want to continue?" "n"
if [ $? == 0 ]
then
    echo "User wants to continue"
else
    echo "User doesn't want to continue"
fi
```

8.4.363 Quality and Security Recommendations

1. Validate that $1 (the prompt text) exists and is non-empty before proceeding.
2. Regularly audit and update third-party dependencies to keep them up to date with the latest security updates.
3. Avoid using raw user inputs without validation. In this case though, the input is controlled to "y" or "n" only.
4. Use case-insensitive matching to allow inputs as 'Y', 'n', etc.

5. Provide more detailed information about valid inputs for prompt. Do not assume users know they need to enter 'y' or 'n'.
6. Test the function rigorously with a variety of different inputs and edge cases to ensure it handles those correctly.
7. Consider implementing a limit on the number of invalid attempts before automatically selecting the default option.
8. Follow secure code practices and maintain a regular schedule for reviewing/updating the function.

8.4.364 `url_decode`

Contained in `lib/functions.d/hps_log.sh`

Function signature: ff3afbc0000d42d9f1561eebfaa989db874faafa82b7e2cdf10838044a3f376d

8.4.365 Function overview

The `url_decode()` function is a bash function which decodes a URL-encoded string. The function is part of a logging system that decodes a received message before sending it to the system's log and writing it to a file, if possible. The `url_decode()` function replaces "+" characters in the encoded string with spaces, and then replaces any remaining percent-encoded characters with the corresponding ASCII characters.

8.4.366 Technical description

- **Name:** url_decode
- **Description:** This function takes in a URL-encoded string and decodes it into a usable string format. The decoded message is then passed on to the system's log and written to a file, if possible.
- **Globals:**
  - `VAR: desc`: No global variables are explicitly used within this function.
- **Arguments:**
  - `$1: desc`: This is the URL-encoded string to be decoded.
- **Outputs:** Outputs decoded version of the URL-encoded input string.
- **Returns:** Does not return any explicit value.
- **Example usage:**

```
# Declare a URL-encoded string
url_encoded="Hello%20World%21%0A"

# Decode the message
url_decoded=$(url_decode "$url_encoded")
```

8.4.367 Quality and security recommendations

1. Robustness: Check if the URL-encoded string passed to the function is correctly formatted. This can help prevent unexpected behaviour or errors during the decoding process.
2. Error Handling: Improve error handling by implementing a mechanism to inform the user when the writing to the log file fails.
3. Redundancy: The function currently prints error messages to the console in addition to logging them to the system's log. Consider removing this duplication to make the function more efficient.
4. Security: Avoid logging sensitive information. If the function is used in a situation where the messages to be logged include sensitive data, ensure this data is either not logged or is properly obfuscated before logging.

8.4.368 `url_encode`

Contained in `lib/host-scripts.d/common.d/common.sh`

Function signature: 244a7c143b13a17ef50725eef748c82a2bd2df462a35d3727b81937558bae3ec

8.4.369 Function Overview

The `url_encode` function is used to encode URLs in Bash. The function accepts a string as an input and returns a URL-encoded string as output. It scans through each byte of the input string and performs a case-by-case encoding. Specifically, all alphanumeric characters and the special characters .,_ and - are left as they are. Spaces are replaced with '%20' hex value for fast processing, and all other characters are replaced with their respective '%' followed by their hex value.

8.4.370 Technical Description

- **Name**: url_encode
- **Description**: This function is used for URL encoding in Bash. It accepts a string, iterates through its characters and replaces each character based on particular rules.
- **Globals**: N/A
- **Arguments**: $1: input string for url encoding.
- **Outputs**: URL-encoded string.
- **Returns**: 0
- **Example usage**: `s="HELLO World!?/#"     url_encode "$s"`

8.4.371 Quality and Security Recommendations

1. Consider using a well-established library or built-in function for URL encoding if available, as there might be edge cases we are not considering in this implementa-

tion.
2. Make sure you always encode URLs when using them as part of commands or requests, as failure to do so could lead to command injection or other security vulnerabilities.
3. It is essential to write unit tests for this function to make sure it works as expected. Consider testing with all types of characters including alphanumeric and special characters.
4. Avoid the use of the variables with generic names like 's', 'out', 'c', 'i' to improve readability.
5. Consider handling the case of receiving more than one parameter by showing an error message.
6. Document the purpose of disabling SC2039 ShellCheck warning at the beginning of the function.

8.4.372 `urlencode`

Contained in `lib/functions.d/cgi-functions.sh`

Function signature: f758d39e7a343eef82fc4e92ae1358118cf9a79d8accf9f5013313b5448282ac

8.4.373 Function Overview

The `urlencode` function is a utility function used to encode a string by converting certain characters into their hexadecimal representation prefixed by `%`. The function operates by iterating over each character in the source string and checking if it falls within certain ranges. If it doesn't, that character is replaced by its encoded form.

8.4.374 Technical Description

- **Name:** `urlencode`
- **Description:** The `urlencode` function is designed to encode a string by replacing certain characters with their URL-encoded form based on the ASCII character set. For every character not in the set `[a-zA-Z0-9.~_-]`, it is replaced with its hexadecimal ASCII value prefixed by `%`.
- **Globals:** None
- **Arguments:**
  - `$1`: This is the string that needs to be URL-encoded.
- **Outputs:** The function prints the URL-encoded version of the input string.
- **Returns:** None
- **Example Usage:**

```
$ urlencode "Hello, World!"
Hello%2C%20World%21
```

8.4.375 Quality and Security Recommendations

1. Validate the inputs: Before processing, validate that the input is in fact a string and not any other data type to avoid errors during and unexpected results from processing.
2. Error handling: The function currently lacks error handling. Add an error message or an error code to handle situations where an invalid input is given.
3. Unit testing: Ensure each piece of this function is adequately tested, both with expected and unexpected inputs to ensure it behaves as expected in all scenarios.
4. Use of `local`: This function appropriately uses `local` variables to ensure that they do not clash with variables outside of the function. This should be continued for any new variables introduced into the function.
5. Security: The function as is does not have any direct security concerns. However, always be aware of potential security risks when dealing with URL encoding, especially in web development contexts where URL encoded strings can sometimes be manipulated by malicious actors.

8.4.376 `verify_required_repo_packages`

Contained in `lib/functions.d/repo-functions.sh`

Function signature: a16392408f8b201975834d3d38029e3859302d45f825b35156dca0ae85e71609

8.4.377 Function Overview

The `verify_required_repo_packages` function checks for the presence of certain required packages in a specified repository path. It is intended for use with RPM-based package repositories. The function takes the repo path as the first argument, then a list of required package names. If the function cannot find a required package in the repository, it logs an error message and returns a value of 1 or 2, depending on the type of error. If all required packages are present, it logs an informational message and returns a zero value.

8.4.378 Technical Description

- **Name**: `verify_required_repo_packages`
- **Description**: This function inspects a specified package repository and verifies the presence of required packages. It is used for checking the availability of certain essential packages in a RPM repository.
- **Globals**: None.
- **Arguments**:
  - `$1: repo_path` - The path to the package repository.
  - `$2...: required_packages` - An array of package names that the function will check for in the repository.

- **Outputs**: Logs error messages through `hps_log` function if required packages are missing or repository path is not provided. Logs an informational message if all required packages are present.
- **Returns**:
  - `0` if all required packages are present in the specified repo_path.
  - `1` if repo_path not provided or does not exist.
  - `2` if any of the required packages are missing.
- **Example Usage**: `verify_required_repo_packages "${HPS_PACKAGES_DIR}/${DIST_STRING}/R` `zfs opensvc`

8.4.379 Quality and Security Recommendations

1. It is recommended to use absolute paths for `repo_path` to avoid any ambiguity or errors derived from relative paths usage.
2. Ensure the proper access rights are in place for the directory path specified by `repo_path`, so the function can process the commands effectively.
3. Always sanitize input given to the function to prevent potential security vulnerabilities, such as command injection.
4. Consider adding further error checking mechanisms, like checking if each package name in `required_packages` is a non-empty, non-null string.
5. Implement a feature to handle version-specific packages in the array `required_packages`. Currently, it assumes that the requirement is met if any version of the package exists.
6. Leverage the return status of the function to handle error situations in the script that calls this function.

8.4.380 `zfs_get_defaults`

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: 7075829ac859fa1a3d095289b6f87eeae5bfd9df08c9d1cfd66df7de132cf128

8.4.381 Function Overview

The function `zfs_get_defaults` is designed for setting sensible defaults for pool options (`_POOL_OPTS`) and ZFS properties (`_ZFS_PROPS`). This function helps to customize and optimize your ZFS filesystem according to your needs. Pool options include sector size, while ZFS properties cover settings such as compression type, access time, extended attribute style, Access Control List (ACL) type, mode, inheritance, node size, and log bias.

8.4.382 Technical Description

- Name: `zfs_get_defaults`

- Description: This function sets default values for pool options (`_POOL_OPTS`) and ZFS properties (`_ZFS_PROPS`). The pool options are safest for SSD/NVMe/HDD with 4K-sectors. The ZFS properties include features like compression, access time, ACLs, and others.
- Globals:
  - `_POOL_OPTS`: An array to store pool options.
  - `_ZFS_PROPS`: An array to store ZFS properties.
- Arguments:
  - `$1`: A reference to a variable intended to hold pool options.
  - `$2`: A reference to a variable intended to hold ZFS properties.
- Outputs: There are no explicit outputs besides the modified `_POOL_OPTS` and `_ZFS_PROPS` variables.
- Returns: This function does not have any explicit return values and does not generate exit status.
- Example usage: `zfs_get_defaults POOL_OPTS ZFS_PROPS`

8.4.383 Quality And Security Recommendations

1. Consider making the function's name more descriptive, such as `set_zfs_defaults`, to specify the action that the function performs.
2. For improved security, validate the variables that are passed into the function to ensure they allow item assignment.
3. When setting sensible defaults, remember to base it on the actual use case scenario and the nature of the data handled.
4. To enhance transparency and ease of debugging, consider implementing a logging system to record any changes made by the function.
5. The comments flagged with 'TODO' should be addressed. Consider implementing the -O props within the function as these can provide an additional level of customization for the user.
6. Always include error checking in your functions to catch unexpected scenarios and improve robustness.

8.4.384 `zpool_create_on_free_disk`

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: ff0bf00dc7c2ed8816d8a88a6b148a5993ba6e4c5d1ae70415c5960612576a80

8.4.385 1. Function overview

The `zpool_create_on_free_disk` function is a Bash utility for working with data storage in a Unix-like operating system. It initializes a zpool on free disk space using a specified strategy. It has preset values for variables such as `strategy` (defaults to "first"), `mpoint` (defaults to "/srv/storage"), `force`, `dry_run`, and `apply_default` all of which can be adjusted according to user requirements.

8.4.386 2. Technical description

- **Name**: zpool_create_on_free_disk
- **Description**: A bash function designed to initialize a zpool on available disk space. This function allows users to set their preference with a number of preset options to customize the setup according to their needs. The possible options include specifying a storage strategy, the mount point for the storage, and whether to force the operation, do a dry run or apply default settings.
- **Globals**:
  - `strategy`: Defines the strategy for initializing the zpool storage. Default is "first".
  - `mpoint`: The directory in which the initialized storage will be mounted. Default is "/srv/storage".
  - `force`: Defines whether or not to force the initialization. Default is 0 (do not force).
  - `dry_run`: Defines whether or not to perform a dry run. Default is 0 (do not dry run).
  - `apply_defaults`: Defines whether to apply the default settings. Default is 1 (apply defaults).
- **Arguments**:
  - `$1`: The first positional parameter is not explicitly used in this function.
  - `$2`: The second positional parameter is not explicitly used in this function.
- **Outputs**: Outputs the status of the zpool creation process.
- **Returns**: Returns a status code indicating the success or failure of creation.
- **Example usage**: To use this function, you would typically include in a Bash script like this:

```
zpool_create_on_free_disk
```

8.4.387 3. Quality and security recommendations

1. Validate input: Although this function does not take arguments, it is always good practice to ensure any input or sources from which input is derived are valid.
2. Error handling: Include error handling mechanisms for situations where the disk space is not available or the formatted storage cannot be mounted at the specified mount point.
3. Security: Ensure that the storage's mount point has appropriate permissions set, and sensitive data is securely managed.
4. Code clarity: Some variables are initialized but not used, these could be removed for improved code clarity.

5. Logging: Add comprehensive logging for tracking the sequence of operations. Logs would aid in debugging and resolving any issues that might arise.
6. Code Comments: Adding comments to explain the purpose of complex code blocks would make the function more maintainable.

8.4.388 `zpool_name_generate`

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: 5995908b1c71b7b0931db1a09cf94c2257d6d0ed783b7e45ca8926f62b975cf6

8.4.389 Function Overview

The `zpool_name_generate` function is a Bash function that generates a ZFS pool (zpool) name based on certain parameters. It specifically takes an input, "class," and produces a zpool name incorporating information about the type of storage (like `NVMe`, `SSD`, HDD, ARC, or `MIX`), the cluster name, current Unix timestamp, and a random 3-byte hexadecimal identifier.

8.4.390 Technical Description

- **Name**: `zpool_name_generate`
- **Description**: This function receives a "class" parameter, which represents the type of storage, and generates a unique zpool name that includes the storage type, cluster name, Unix timestamp, and a random hexadecimal identifier.
- **Globals**: None
- **Arguments**:
  - `$1 (class)`: an indicator of the storage class. It could be either of `nvme`, `ssd`, hdd, `arc`, `mix`. If it's not set or doesn't match these, the function will return an error.
- **Outputs**: This function outputs a zpool name to stdout.
- **Returns**:
  - Return code 2 if the class argument is not given or doesn't match the expected values.
  - Return code of the `zpool_slug` function call if it fails.
- **Example Usage**:

```
$ zpool_name_generate nvme
```

8.4.391 Quality and Security Recommendations

1. The function doesn't validate the cluster name from the `remote_cluster_variable` function. Such validation could be beneficial to avoid creating zpools with incorrect or malicious names.
2. The random three-byte value is generated using both `/dev/urandom` and the `RANDOM` variable. To maintain consistency, consider using only one of these methods. Using `/dev/urandom` would make it more random and secure.
3. Consider adding error checking for critical operations like od and `printf`, which could fail due to various reasons.

8.4.392 `zpool_slug`

Contained in `lib/host-scripts.d/common.d/zpool-management.sh`

Function signature: 18bbbffc19d9426de745e5b45fe837f9d50ec9443122924bb19421975f877cbc

8.4.393 Function overview

The `zpool_slug()` function is designed to convert a given string into a slug, suitable for use in URL paths or IDs. This function takes up to two arguments: the string to convert and (optionally) the maximum length of the slug. It converts the string to lower case, replaces any non-alphanumeric characters with a hyphen and removes any consecutive or trailing hyphens. It then truncates the slug at the specified maximum length or at 12 characters if no length was provided.

8.4.394 Technical description

- **name**: `zpool_slug()`
- **description**: Converts a string into a slug of a specified or default length.
- **globals**: None.
- **arguments**: [`$1`: The string to be transformed into a slug, `$2`: Maximum length of the slug. This argument is optional, with a default length of 12 if left unspecified.]
- **outputs**: Prints the slug to stdout
- **returns**: None.
- **example usage**: `zpool_slug "Example String" 10` will output `example-str`

8.4.395 Quality and security recommendations

1. Right now, there is no explicit handling of invalid input, such as non-string values. Adding type checking and error handling for these scenarios would improve robustness.

2. Consider adding input sanitization to prevent any potential security issues (although current implementation is already reasonably safe due to removal of all non-alphanumeric characters).
3. Ensure that maximum slug length is not overly restrictive and take possible unicode characters into account.
4. Write unit tests for this function to guarantee it behaves as expected and validate the slug creation logic.
5. Specify locale in the script to ensure consistent character conversion, as the current implementation uses the locale setting of the running environment which can lead to unexpected results.