Secure Deployment and Disposal

Before deploying our system it is important to perform system hardening procedures and establish secure disposal in order to ensure the safety and security of our system during and at the end of its lifetime.

System hardening

Systems hardening is a collection of tools, techniques, and best practices to reduce vulnerability in technology applications, systems, infrastructure, firmware, and other areas. The goal of systems hardening is to reduce security risk by eliminating potential attack vectors and condensing the system's attack surface.

The "attack surface" is the combination of all the potential flaws and backdoors in technology that can be exploited by hackers. These vulnerabilities can occur in multiple ways, including:

- Default and hardcoded passwords
- Passwords and other credentials stored in plain text files
- Unpatched software and firmware vulnerabilities
- Poorly configured BIOS, firewalls, ports, servers, switches, routers, or other parts of the infrastructure
- Unencrypted network traffic or data at rest
- Lack, or deficiency, of privileged access controls

The best practices for system hardening are:

- *Auditing our existing system* - carrying out a comprehensive audit of our existing technology using penetration testing, vulnerability scanning, configuration management and other tools to find flaws in our system and prioritize fixes. After that, conduct system hardening assessments against resources using industry standards from CIS, NIST, DISA, etc.
- *Create a strategy for systems hardening* - instead of carrying out comprehensive system hardening on our entire system all at once, create a strategy and plan based on risks identified in our system and remediate the biggest flaws first.
- *Patch vulnerabilities immediately* - ensuring an automated and comprehensive vulnerability identification and patching system.

- *Network hardening* - ensure our firewall is properly configured and that all rules are regularly audited, secure remote access points and users, block any unused or unneeded open network ports, disable and remove unnecessary protocols and services, implement access lists and encrypt network traffic
- *Server hardening* - putting all our servers in a secure datacenter, never test hardening on production servers, always harden servers before connecting them to the internet or external networks, avoid installing unnecessary software on a server, segregate servers appropriately, ensure superuser and administrative shares are properly set up, and that rights and access are limited in line with the principle of least privilege.
- *Application hardening* - Remove any components or functions we do not need, restrict access to applications based on user roles and context (such as with application control), remove all sample files and default passwords. Application passwords should then be managed via an application password management/privileged password management solution, that enforces password best practices (password rotation, length, etc.). Hardening of applications should also entail inspecting integrations with other applications and systems, and removing, or reducing, unnecessary integration components and privileges.
- *Database hardening* - create admin restrictions, such as by controlling privileged access, on what users can do in a database, turn on node checking to verify applications and users, encrypt database information - both in transit and at rest, enforce secure passwords, introduce role-based access control (RBAC) privileges and remove unused accounts.
- *Operating system hardening* - apply OS updates, service packs, and patches automatically, remove unnecessary drivers, file sharing, libraries, software, services, and functionality, encrypt local storage, tighten registry and other systems permissions, log all activity, errors, and warnings, implement privileged user controls.
- *Eliminate unnecessary accounts and privileges* - enforce least privilege by removing unnecessary accounts (such as orphaned accounts and unused accounts) and privileges throughout our system.

Benefits of system hardening are substantial, and they include:

- *Enhanced system functionality* - since fewer programs and less functionality means there is less risk of operational issues, misconfigurations, incompatibilities, and compromise.
- *Significantly improved security* - a reduced attack surface translates into a lower risk of data breaches, unauthorized access, systems hacking, or malware.
- *Simplified compliance and auditability* - fewer programs and accounts coupled with a less complex environment means auditing the environment will usually be more transparent and straightforward.

Secure disposal

Control 7.14 or ISO 27002:2022 is a preventive type of control that requires organisations to maintain the confidentiality of information hosted on the equipment that will be disposed of or deployed to be reused by establishing and applying appropriate procedures and measures for disposal and reuse.

We will be using its principles at the end of our systems lifecycle.

Compliance with Control 7.14 requires the establishment of an organisation-wide data disposal-reuse procedure, identification of all equipment and implementing suitable technical disposal mechanisms and reuse processes.

There are 4 key considerations that need to be taken into account for compliance with Control 7.14:

- *A proactive approach should be adopted* - before disposal takes place or the equipment is made available for reuse, we must confirm whether the equipment contains any information assets and licensed software and should ensure that such information or software is permanently deleted.
- *Physical destruction or irretrievable deletion of information* - there are 2 methods of doing this in Control 7.14:
    - Equipment hosting storage media devices that contain information should be physically destroyed.
    - Information stored on the equipment should be erased, overwritten, or destroyed in a non-retrievable manner so that malicious parties cannot access information. We should look into Control 7.10 on Storage Media and Control 8.10 on the deletion of Information.

- *Removal of all labels and markings* - components of the equipment and the information contained in it can have labels and markings that identify the organisation or that disclose the name of the asset owner, network, or information classification level assigned. All these labels and markings should be irretrievably destroyed.
- *Removal of controls* - uninstall all security controls such as access restrictions or surveillance systems when they vacate facilities:
    - The terms of the lease agreement related to conditions on which it needs to be returned.
    - Eliminating and mitigating the risk of unauthorised access to sensitive information by the next tenant.
    - Whether the existing controls can be reused at the next facility.

There is also 3 more thing to take into account however, and those are:

- *Damaged equipment* - when sending damaged equipment for repair, the information it contains might be obtained by unwanted third parties, therefore it should be considered whether to destroy it instead of repairing it.
- *Full disk encryption* - maintain the confidentiality of cryptographic keys. For example, the encryption key should not be stored on the same disk.
- *Overwriting tools* - we should choose an overwriting technique taking into account:
    - Level of information classification assigned to the information asset.
    - Type of storage media on which the information is stored.

Secure deployment

Small, regular code commits can automatically trigger builds and run comprehensive testing. Whole system deployments can be made to development and reference environments, prior to deploying an identical production environment. The solution for this time consuming task is a deployment pipeline that minimises the need for manual processes, allowing fully-tested regular production deployments in a matter of minutes.

The security of this process is critical in order to protect the integrity of our code and the systems it builds. The build and deployment pipeline needs to maintain the integrity of your code while it's being deployed.

The steps:

- *Using a trusted pipeline* - consider how the pipeline itself is administered and managed, including the underlying infrastructure. If the security of these components is compromised, it is difficult to assert trust in code you build and deploy through it. If the pipeline is managed by a service provider, use the NCSC Cloud Security Principles to understand the security properties of their service. Layering a process of cryptographic signing and code verification on top of your repository can help to increase confidence that the code has not been tampered with.
- *Peer review code before deployment* - accept or reject according to our development processes. This step is an important source of quality control, so we should implement technical controls to prevent it being bypassed.
- *Control how deployments are triggered* - code that triggers automated deployment to production environments should be carefully managed. Production deployments shouldn't be made from untrusted code repositories, forks or branches.
- *Run automatic testing as part of our deployments* - where possible, automate testing so that it supports the deployment pipeline. 'Cheap' tests can be run on every code change, resource-intensive tests can be saved for notable releases. Don't ignore tests that fail. Re-evaluate and refactor testing that does not work for us.
- *Carefully manage secrets and credentials* - care in how we store, manage and inject these into your environment. Rotating keys can be used and consider how this process can be automated to make it practical in our workflow.
- *Ensure deployment pipeline controls cannot be bypassed, or re-ordered* - all deployment processes should be followed from development through to deployment. This could be achieved architecturally, or by carrying out cryptographic signing and verification at each stage.
- *Avoid 'self policing'* - the pipeline should enforce rules that define whether code is accepted or rejected before deployment. It's possible that these controls are themselves implemented in source code, along with your product. It should not be possible for individuals to 'police themselves' by modifying these rules.
- *Be cautious of importing third party libraries and their updates* - ensure only intended dependencies are included and that they come from legitimate sources.
- *Consider hard breaks and approval* - fully automated deployment without adequate security controls in place is high risk. If you don't

have confidence in the deployment pipeline processes, consider a hard break that requires approval before releases go live in a production environment.

- *Keep track of the repository* - monitor all branches and pull requests, such that unwanted attackers are not able to attack the system through its automated process.