Secure By Design (SBD) Principles

Brunel University London

An ISO/IEC 27001:2013: *Aligned Document - Implementing Cyber and Information Security Best Practice*

Mick Jenkins

Chief Information Security Officer

Document History

Version	Author	Comments	Date
V 0.1	Andrew Clarke	Initial Draft	03/06/2019
V 1.0	Andrew Clarke	Approved CISO, CIO & DPO	20/06/2019
	Andrew Clarke	Annual Review	08/06/2020

Document Approval

The contents of this document are classified as Protect to Brunel University London (BUL) information classification. Proprietary information presented in this document may not be used without written consent from BUL and remains the exclusive property of BUL unless otherwise agreed to in writing.

Owner: Michael Jenkins	Chief Information Security Officer
Signature: MGJ	Date: 20 Jun 2019
Approver: Pekka Kahkipuro	Chief Information Officer
Signature: PK	Date: 20 Jun 2019
Distribution:	

This document requires the approval from BUL as defined in the ISMS Compliance document.

Contents

1. About this document

1.1 Purpose of Document

Brunel University is committed to safeguarding its information and computing infrastructure upon which the teaching and research functions rely. Additionally, the University is strongly committed to maintaining the security and privacy of confidential personal information and other data it collects or stores.

In order to guide the University community in achieving these objectives, the University adheres to the Secure By Design (SBD) philosophy.

Secure By Design, in IS terms, means that the deployment of both hardware (infrastructure architecture) and services (software architecture) has been designed from the foundation to be secure.

Security by Design is an approach to software and hardware development that seeks to make systems as free of vulnerabilities and impervious to attack as possible through such measures as standard secure builds, continuous testing, authentication safeguards and adherence to best programming practices.

The University has a formal Vulnerability Management Programme (VMP) to bring in industry good practice set against our risk management process to maintain the SBD principles.

Please refer to Brunel University London ISMS Document BUL-GLOS-000 - SyOPs Glossary of Terms for the glossary of terms, acronyms and their definitions for the suite of Brunel University London ISMS documentations.

1.2 Responsibilities

Table 1 – responsibilities

Title / Role	Description
All	Responsible for adhering to the SBD principles

1.3 ISO 27001 - Conformance

This section indicates the University Conformance to ISO27001:2013.

University ISMS Control Number	SOA – Number A14 – System acquisition, development and maintenance
ISO 27001:2013 Conformance Control	Information Classification Objective A.14.1.1 Information security requirements analysis and specification

1.4 Scope

The scope of these Principles apply to:

- Any server or client that IS manages or is responsible for, including servers which are managed by third parties on behalf of IS.

- Any server or client that College IT manages or are responsible for, including servers which are managed by third parties on behalf of Colleges.

- Any software on these servers or clients. In this document, “software” shall be taken to include firmware, BIOS, hypervisor, operating system, driver, library, middleware, application, service, and other digital capabilities.

- All public-facing Cloud systems and services that the University subscribes to including PaaS, SaaS, and IaaS.

1.5 Principles Maintenance

Supporting standards, guidelines and procedures will be issued on an ongoing basis by Brunel University London. Users will be informed of any subsequent changes or updated versions of such standards, guidelines and procedures by way of e-mail or other relevant communication media. Users shall then have the obligation to obtain the current information systems policies from Brunel University London intranet (IB) or other relevant communication media on an ongoing basis and accept the terms and conditions contained therein.

1.6 References

BUL Change Control Process (ISMS 12.1.2)

BUL-POL-12.6 - Vulnerability Management

BUL-POL-12.6 - Patch Management

BUL-PR-14.08 - Privacy By Design Principles

2.0 Secure By Design Principles

Architects and solution providers need guidance to produce secure systems, services and applications by design, and they can do this by not only implementing the basic controls documented in the main text, but also referring back to the underlying “Why?” in these principles.

Core pillars of information security:
Information security has relied upon the following pillars:

- Confidentiality – only allow access to data for which the user is permitted
- Integrity – ensure data is not tampered or altered by unauthorized users
- Availability – ensure systems and data are available to authorized users when they need it

The following principles are all related to these three pillars. Indeed, when considering how to construct a control, considering each pillar in turn will assist in producing a robust security control. The systems, services and applications will be the more robust the more principles applied to them.

For example, it is a fine thing when implementing data validation to include a centralized validation routine for all form input. However, it is a far finer thing to see validation at each tier for all user input, coupled with appropriate error handling and robust access control.

Security architecture

Hardware (infrastructure architecture) and services (software architecture) architects are responsible for constructing their design to adequately cover risks from both typical usage, and from extreme attack. Architects and designers must cope with extreme events, such as brute force or injection attacks, and fraud. The risks for architects and solution designers are well known. The days of “we didn’t know” are long gone. Security is now expected, not an expensive add-on or simply left out.

Security architecture refers to the fundamental pillars: the application must provide controls to protect the confidentiality of information, integrity of data, and provide access to the data when it is required (availability) – and only to the right users.
When starting a new application or system or re-factoring an existing application or system, consider each functional feature, and consider:

- Is the process surrounding this feature as safe as possible? In other words, is this a flawed process?
- If I were evil, how would I abuse this feature?
- Is the feature required to be on by default? If so, are there limits or options that could help reduce the risk from this feature?

The best system architecture designs and detailed design documents contain security discussion in each and every feature, how the risks are going to be mitigated, and what was actually done during coding.

Security architecture starts on the day the business requirements are modelled, and never finishes until the last copy of your application is decommissioned. Security is a life-long process, not a one shot accident.
Security principles

Asset classification is key to the SBD principles: Selection of controls is only possible after classifying the data to be protected. For example, controls applicable to low value systems such as blogs and forums are different to the level and number of controls suitable for intellectual property, student personal information, and electronic finance systems.

About attackers

When designing controls to prevent misuse of University applications and systems, consider the most likely attackers (in order of likelihood and actualised loss from most to least):

- Disgruntled staff or developers
- “Drive by” attacks, such as side effects or direct consequences of a virus, worm, or Trojan attack
- Motivated criminal attackers, such as organised crime
- Criminal attackers without motive against the University, such as defacers
- Script kiddies and hackers
- Fix security issues correctly

Once a security issue has been identified, it is important to develop a test for it, and to understand the root cause of the issue. When design patterns are used, it is likely that the security issue is widespread amongst all code bases, so developing the right fix without introducing regressions is essential.
For example, a user has found that they can see another user’s balance by adjusting their cookie. The fix seems to be relatively straightforward, but as the cookie handling code is shared among all applications, a change to just one application will trickle through to all other applications. The fix must therefore be tested on all affected applications.

2.1 Fourteen principles

1. Secure the weakest link
2. Defend in depth
3. Fail secure
4. Grant least privilege
5. Separation of Duties
6. Minimise Surface Area
7. Economise mechanism (Keep security simple and usable)
8. Authenticate requests
9. Control access
10. Least Common Mechanism
11. Open Design
12. Avoid Security by Obscurity
13. Don’t Trust Third Parties
14. Production Acceptance

1. **Principle of secure the weakest link**
   - Security practitioners often point out that security is a chain; and just as a chain is only as strong as the weakest link, a software security system is only as secure as its weakest component.

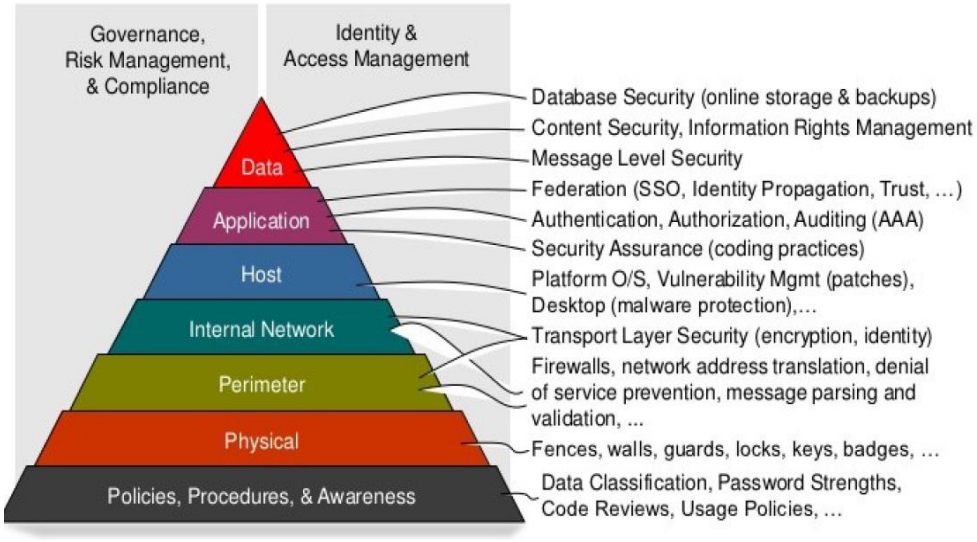
   - Attackers go after the weakest point in a system, and the weakest point is rarely a security feature or function. When it comes to secure design, make sure to consider the weakest link in your system and ensure that it is secure enough.

   - The weakest link principle requires that in designing security for a system, focus should be put on the weakest components of the overall system, because just as the old saying goes, “a chain is only as strong as its weakest link”.

2. **Principle of Defense in Depth**
   - The principle of defense in depth suggests that where one control would be reasonable, more controls that approach risks in different fashions are better. Controls, when used in depth, can make severe vulnerabilities extraordinarily difficult to exploit and thus unlikely to occur.

   - With secure coding, this may take the form of tier-based validation, centralized auditing controls, and requiring users to be logged on all pages.
     Example, a flawed administrative interface is unlikely to be vulnerable to anonymous attack if it correctly gates access to production management networks, checks for administrative user authorization, and logs all access.

   - Redundancy and layering - Also called the "belt and braces" approach.
     - Redundancy and layering is usually a good thing in security. Don’t count on the firewall to block all malicious traffic; use an intrusion detection system (IDS) as well. When designing an application, prevent single points of failure with security redundancies and layers of defence
     - The idea behind defence in depth is to manage risk with diverse defensive strategies, so that if one layer of defence turns out to be inadequate, another layer of defence will hopefully prevent a full breach.



Defense in Depth examples:

- Anti-virus software
- Authentication and password security
- Biometrics
- Demilitarized zones (DMZ)
- Encryption
- Firewalls (hardware or software)
- Hashing passwords
- Intrusion detection systems (IDS)
- Logging and auditing
- Multi-factor authentication
- Vulnerability scanners
- Physical security (e.g. smart access control doors)
- Timed access control
- Internet Security Awareness Training
- Virtual private network (VPN)
- Sandboxing
- Intrusion Protection System (IPS)

3. Principle of Fail secure

- A fail-secure system is one that, in the event of a specific type of failure, responds in a way such that access or data are denied.
  - ➢ Related: a fail-safe system, in the event of failure, causes no harm, or at least a minimum of harm, to other systems or to personnel.
- Fail-secure and fail-safe may suggest different outcomes.

- For example, if a building catches fire, fail-safe systems would unlock doors to ensure quick escape and allow firefighters inside, while fail-secure would lock doors to prevent unauthorized access to the building

Fail secure

- Default is to deny access.
- Developers should check return values for exceptions/failures.
- Base access decisions on permission rather than exclusion. This principle means that the default situation is lack of access, and the protection scheme identifies conditions under which access is permitted.

The alternative, in which mechanisms attempt to identify conditions under which access should be refused, presents the wrong psychological base for secure system design.

Applications regularly fail to process transactions for many reasons. How they fail can determine if an application is secure or not.

Fail-insecure concepts

- Fail-insecure often introduced by desire to support legacy (insecure) versions. e.g.TLS allows old clients to use weak crypto keys
- Sometimes introduced because it may be easier to work with short “blacklist” than with long “whitelist”.

4. Principle of Least privilege

Also known as the principle of minimal privilege or the principle of least authority.

The principle of least privilege (PoLP) recommends that accounts have the least amount of privilege required to perform their business processes. This encompasses user rights, resource permissions such as CPU limits, memory, network, and file system permissions.

Requires that in a particular abstraction layer of a computing environment, every module (such as a process, a user, or a program, depending on the subject) must be able to access only the information and resources that are necessary for its legitimate purpose.

Rationale

When code is limited in the system-wide actions it may perform, vulnerabilities in one application cannot be used to exploit the rest of the system.

- Example, Microsoft states “Running in standard user mode gives customers increased protection against inadvertent system-level damage caused by malware, such as root. kits, spyware, and undetected viruses”
- Example, if a middleware server only requires access to the network, read access to a database table, and the ability to write to a log, this describes all the permissions that should

be granted. Under no circumstances should the middleware be granted administrative privileges.

- Example, A user account gets only those privileges which are essential to that user's work. A backup user does not need to install software: hence, the backup user has rights only to run backup and backup-related applications. Any other privileges, such as installing new software, are blocked.

- Example, UNIX systems, root privileges are necessary to bind a program to a port number less than 102, to run a mail server on port 25, the traditional SMTP port, a program needs the privileges of the root user. Once set up, the program should relinquish its root privileges, not continue running as root. A large problem with many e-mail and other servers is that they don't give up their root permissions once they grab the mail port (Sendmail is a classic example).

Often least privilege assignment fails owing to:

- It's an effort to figure out what the least privilege needed actually is.
- The lazy (or overworked) designer or programmer will often assign the max privilege, because that's easy.
- "If we don't run as admin, stuff breaks"
- It's very hard to design systems that don't have some root/sysadmin that has all the privileges

Separate privileges

- Design systems with many different privileges
- “file access” is not one privilege, but many
- “network access”: same thing
- Reading vs writing– User/smart card interface

Separation of Privileges

Split system into pieces, each with limited privileges.

Implementation in software engineering:

Have computer program fork into two processes.

• The main program drops privileges (e.g. dropping root under Unix).

• The smaller program keeps privileges in order to perform a certain task.

• The two halves then communicate via a socket pair.

Benefits:

• A successful attack against the larger program will gain minimal access, even though the pair of programs will perform privileged operations.

5. Principle of Separation of duties

A key fraud control is separation of duties. For example, someone who requests a computer cannot also sign for it, nor should they directly receive the computer. This prevents the user from requesting many computers, and claiming they never arrived.
Certain roles have different levels of trust than normal users. In particular, administrators are different to normal users. In general, administrators should not be users of the application.
For example, an administrator should be able to turn the system on or off, set password policy but shouldn't be able to log on to the storefront as a super privileged user, such as being able to "buy" goods on behalf of other users.

Segregation of Duties

Achieved by:

- Functional separation:

Several people should cooperate. Examples:

- one developer should not work alone on a critical application,
- the tester should not be the same person as the developer
- If two or more steps are required to perform a critical function, at least two different people should perform them, etc. This principle makes it very hard for one person to compromise the security, on purpose of inadvertently.

- Dual Control:
  - Example 1: in the SWIFT banking data management system there are two security officers: left security officer and right security officer. Both must cooperate to allow certain operations.
  - Example 2: nuclear devices command.
  - Example 3: cryptographic secret sharing

6. Principle of Minimise attack surface area

Every feature that is added to an application adds a certain amount of risk to the overall application. The aim for secure development is to reduce the overall risk by reducing the attack surface area.
For example, a web application implements online help with a search function. The search function may be vulnerable to SQL injection attacks. If the help feature was limited to authorized users, the attack likelihood is reduced. If the help feature's search function was gated through centralized data validation routines, the ability to perform SQL injection is dramatically reduced. However, if the help feature was re-written to eliminate the search function (through better user interface, for

example), this almost eliminates the attack surface area, even if the help feature was available to the Internet at large.

7. Principle of Keep security simple (Economising mechanism)

Attack surface area and simplicity go hand in hand. Certain software engineering fads prefer overly complex approaches to what would otherwise be relatively straightforward and simple code. Developers should avoid the use of double negatives and complex architectures when a simpler approach would be faster and simpler.
For example, although it might be fashionable to have a slew of singleton entity beans running on a separate middleware server, it is more secure and faster to simply use global variables with an appropriate mutex mechanism to protect against race conditions.

Why things get complex

- Desire for features; mission-creep
- New technologies, new possibilities
- Stretches the designs we already have
- Causes interfaces to get used in ways not intended
- Causes new stuff to be layered on top of old stuff
- Desire for legacy compatibility
- Desire to keep things interoperable

Complexity leads to insecurity

If your security mechanisms are too annoying and painful, your users will go to great length to circumvent or avoid them. Make sure that your security system is as secure as it needs to be, but no more.

If you affect usability too deeply, nobody will use your stuff, no matter how secure it is. Then it will be very secure, and very near useless

Don’t frustrate people.

- Minimise the number of clicks
- Minimise the number of things to remember
- Make security easy to understand and self-explanatory
- Security should NOT impact users that obey the rules.

Established defaults should be reasonable.

- People should not feel trapped.

It should be easy to

- Restrict access

- Give access
- Personalise settings

Psychological Acceptability

Psychological acceptability refers to the ease of use of the user interface of any security control mechanism such as authentication, password reset, password complexity, etc. The more user friendly the interface is, the less likely the user will make a mistake in using the security control and expose the system to security breaches.

Establish secure defaults
There are many ways to deliver an “out of the box” experience for users. However, by default, the experience should be secure, and it should be up to the user to reduce their security with simplicity – when permitted.
For example, by default, password aging and complexity should be enabled. Users might be allowed to turn these two features off to simplify their use of the application and increase their risk.

8. Principle of Authenticate requests

Be reluctant to trust

- Assume that the environment where the system operates is hostile. Don’t let just anyone call APIs, and certainly don’t let just anyone gain access to University information.
- If reliant on a cloud component, put in checks to make sure that it has not been spoofed or otherwise compromised.
- Anticipate attacks such as command-injection, cross-site scripting, and so on.

Be Reluctant to Trust Cf. Least Privilege.

Trust Definitions:

- A trusted system [paradoxical definition]: one that can break the security policy.
- Trustworthy system: one that won’t fail us.

What is trusted and not trustworthy?

What is not trusted, but is trustworthy?

Definitions:

- A trusted system [paradoxical definition]: one that can break the security policy.
- Trustworthy system: one that won’t fail us.

An employee who is selling University Intellectual Property is trusted and NOT trustworthy at the same time.

Example OneDrive for Business is trustworthy, and yet a member of staff encrypts University files before putting them there. Then, for this staff member, OneDrive for Business is trustworthy but not trusted.

9. Principle of Control access

Complete Mediation

- Every access and every object should be checked, every time. Make sure the access control system is thorough and designed to work in the multi-threaded world.
- Make sure that if permissions change on the fly in systems, that access is systematically rechecked. Don't cache results that grant authority or wield authority.
- In a world where massively distributed systems are pervasive and machines with multiple processors are the norm, this principle can be tricky to implement.

Complete mediation means that every access to every data object must be checked for identification, authentication, and authorization. This principle forces a system-wide central point of access control. This security principle requires complete access control of every request whether the information system is undergoing initialization, recovery, shutdown, or maintenance.

10. Principle of Least Common Mechanism

The security principle of least common mechanism means to minimize the mechanisms shared by multiple subjects to gain access to University data resource.

➢ Example, serving an application on the Internet allows both attackers and users to share the Internet to gain access to the application. If the attackers launch a DDOS attack and over-load the application, the legitimate users will be unable to access the application.

➢ Example: serving the same login page for your employees, customers, and partners to login to your company portal. The login page thus must be designed to the satisfaction of every user, which is a job presumably harder than having to satisfy only one or a few users. A better design based on the "least common mechanism" principle would be to implement different login pages for different types of users.

Leveraging Existing Components

The security principle of leveraging existing components requires that when introducing new system into the environment, you should review the state and settings of the existing security controls in the environment and ensure that they are being used by the new system to be deployed.

11. Principle of Open Design

The security principle of open design means that security designs that are open to scrutiny and evaluation by the public security community at large are in general more secure than obscure security designs that are proprietary and little known to the public. The rationale behind this principle is that weaknesses in the open designs will have a better chance of been caught and corrected.

12. Principle of Avoid security by obscurity

Security through obscurity is a weak security control, and nearly always fails when it is the only control. This is not to say that keeping secrets is a bad idea, it simply means that the security of key systems should not be reliant upon keeping details hidden.

Example, the security of an application should not rely upon knowledge of the source code being kept secret. The security should rely upon many other factors, including reasonable password policies, defense in depth, business transaction limits, solid network architecture, and fraud and audit controls.

A practical example is Linux. Linux's source code is widely available, and yet when properly secured, Linux is a hardy, secure and robust operating system.

13. Don't trust Third Parties

The University utilises the processing capabilities of third party partners, who more than likely have differing security policies and posture than the University. It is unlikely that the University can influence or control any external third party, whether they are home users or major suppliers or partners.
Therefore, implicit trust of externally run systems is not warranted. All external systems should be treated in a similar fashion.
Example, a loyalty program provider provides data that is used by Finance, providing the number of reward points and a small list of potential redemption items. However, the data should be checked to ensure that it is safe to display to end users, and that the reward points are a positive number, and not improbably large.

14. Production Acceptance

Any new or replacement or any major configuration change to the University services, servers, systems or network architectural must include a vulnerability test before being introduced to the live or production environment.

If the vulnerability test identifies any High or Medium (CVSS scores of >4.9) vulnerabilities, these must be remediated or mitigated prior to acceptance in to a production or live environment.

If these cannot be "closed", then the BUL-PROC-12.6 - Vulnerability Management Exceptions Process must be observed and each vulnerability subject to a risk assessment based upon the CVSS score with reference to the specific and unique impact to the University, probability that the vulnerability can be exploited and the importance of information confidentiality.

If no vulnerability test is conducted, acceptance in to the Live or Production environment will not be accepted.

Secure By Design (SBD) principles are complimented with the Privacy By Design (PBD) principles and ALL new or replacement or any major configuration change to the University services, servers, systems or network architectural must adhere to both sets of principles.