Third Party-Industry Guidance Standards on ‘Secure Coding’

Final Issue 1.0

Date: 19/10/2016,

BT Security

Contents

1. Introduction

1.1. Background

This document lays out general good practice guidance for software development carried out by third-parties supplying applications or systems to BT.

Application layer vulnerabilities are an extremely effective way for people to gain unauthorised access to data or systems. It is increasingly important therefore that the way in which software is developed includes security considerations at its core.

1.2. Objectives

This document attempts to provide a set of guidelines on how to develop software in a way which means that the resulting code will be trusted and secure. There are many aspects to a software development project, and it is a rapidly changing field, so as far as possible a set of high-level principles have been outlined for the different stages of a software project. The document is split into three main sections:

System Design

This section gives guidance for the start of a software project, and outlines several areas which need to be considered before any code is written. By following these steps, it will be easier later on in the project's life to ensure that secure code is delivered.

System Delivery

This section covers the development of the code itself. It covers some general principles (for example source code management), as well as how to avoid some common mistakes/issues that we encounter.

System Assurance:

This section covers the deployment and in-life management of the code you have produced. This may not always be relevant depending on the particular contract, but you should ensure your application or system is able to operate within these guidelines.

In addition, we have included several appendices which outline BT's specifications for Access Control and Cryptography. It is important to make sure that your application is able to comply with these controls

1.3. Additional Materials and Advice

As per the links in the document.

2. System Design

Ref	Control	Reason
1	Data should be classified according to the Data Classification Standard as determined within your organization. This includes design and development information (e.g. designs, issues, requirements) as well as application data. The data must be maintained in accordance with the Information retention policy	By understanding the type of data your system will contain you can put in place the appropriate protections. It's important to secure design and development information (particularly sensitive issues) as unauthorised access could help someone to target the application.
2	All legal & regulatory requirements must be identified.	When writing applications which handle some types of data (e.g. credit card information) there are specific legal/regulatory requirements which will affect how you handle that data.
3	The design process must include a consideration of whether there are security requirements for any people working on the project. (e.g. UK only or if security clearance may be required)	Projects which handle sensitive or government marked information may have requirements about which staff can be used to work on it.
4	Use risk analysis techniques to identify and quantify detailed security risks. Document the risks and proposed mitigations	Each application will have a different set of associated risks (e.g. what environment it is in, what level of data it is handling). It is important to make sure you have considered these, and determined how to protect against them. The risk analysis may be environment dependent, several tools are available to help with this. OWASP has produced a guide to identifying and assessing risks which includes a useful section on how to score them. Alternatively, Microsoft have

		developed a tool to help identify threats to your application, and another to understand your attack surface before and after new apps are deployed If you have any questions or need advice on methodology, contact the standard owner. Note: The technique to use may be project specific, for example IS1 for HMG accredited work
5	Define and document a review schedule of the risks	This will make sure risks are kept current with any changes in the application or its environment. New risks can be added and mitigating action taken
6	Document the Security Architecture to address each risk identified above Provide security guidelines for low level designs / implementation trusted libraries acceptable ciphers acceptable hash algorithms minimum key lengths key management procedures storage segregation / encryption access / transmission of credentials unacceptable protocols	This will make sure you apply appropriate mitigations for the risks you identified, and do so in a coherent way. It will also help design the acceptance tests.
7	Define and document a development lifecycle	Having a defined development lifecycle means that there is a consistent repeatable approach to move from requirements/issues through to a resolution in production. Without this, it can be difficult and time consuming to fix security issues. Microsoft has a model which can be used as a starting point

8	Document and implement the following design procedures: Access control (who can access which portions of the application, and how access is applied for, maintained and revoked) - if this is not managed then it is likely that people will end up with inappropriate access (e.g. users able to perform admin tasks, or retaining access after moving jobs) Clock Synchronisation – if clocks are not accurate it makes it very difficult to use log files to identify issues, especially where you need to correlate with activity on other systems. Data transfers – Data in transit must have protections around it appropriate for its classification How tools and techniques could be used to attack the system - This will allow you to pre-empt attempts and implement mitigations from day-one. Disaster recovery / fallback – This should be clearly defined, and tested, so that the system can react to unexpected events. Document and implement the following development procedures: Include test tools in Continuous Integration – An automated build environment allows each build to be automatically tested for common issues before the application is released. Using static & dynamic security analysis tools as part of the development	Application security is heavily dependent on how it is managed in-life. By defining and implementing the suggested processes you can ensure that the work done in the design and implementation phase is not un-done during the running of the application. Many of these are 'common sense' and will be naturally followed. However, it is important to make the processes explicit to make sure they are followed throughout the life of the application (for example if the support team changes).

	process – This will help to find vulnerabilities early in the development process. Tagging security issues in your source repository and issue tracker – By tagging security bugs it is possible to assess the effectiveness of mitigations/fixes to be assessed. They can also feed back into the risk analysis for the project, and common issues can be shared with other developers to improve general awareness. Using peer review processes – Peer review of code can identify issues which cannot be automatically detected and prevent vulnerabilities caused by coding errors – for example logic errors Deploying from a clean build environment – Using a clean trusted build environment reduces the risk that the output could be polluted with malware, unwanted/old libraries or components which could produce faults of compromise security. Sharing best practice and experience – This can help the community avoid common issues which can lead to vulnerabilities. Document and implement the following in-life management procedures: Timely application of security patches including those for 3rd party components– this will fix security issues which could lead to compromise of your application and its data Hardware & software maintenance - It is important to plan how you will remain on	

	supported hardware and software so you can fix issues and receive security updates Application Start-up/shutdown – An incomplete application shutdown may lead to unexpected behaviour, or artefacts not being cleaned up. Application and job management - The way that the application manages jobs should be defined, especially under unusually heavy loads. Monitoring and alarms – The way that logs and other output is monitored and responded to should be defined and implemented. Archiving/Data cleansing – Data may need to be managed so that it is removed or archived when no longer needed. This can be a requirement of data protection laws. Migration – This is to allow the continuation of the service after an upgrade, for example to fix security issues. Change control – It is important that changes to the application are controlled so that it is possible to understand whether changes are legitimate, and if any issues arise what might have happened to cause them	
9	The security architecture must be reviewed and updated each cycle or iteration. Planned changes to the application must be reviewed to make sure they are not compromising security requirements	This is to make sure it stays up to date with changes in the project's requirements or implementation.

10	Use industry standard security tool kits which are actively maintained, rather than designing your own	Commonly used toolkits are well tested, are less likely to contain major flaws and will be patched quickly if any issues are found. It will also save you time! For example: OpenSSL, OpenSSH, Solaris KCF, Windows Cryptography API, Active Directory, OpenLDAP.
11	Only use components and services required to deliver the products' functionality. Don't expose any interfaces other than where they are required	By minimising unnecessary functionality you reduce the exposure of your application, and can reduce the impact of any bugs which are found.
12	Only use third-party libraries or components which are actively maintained and supported – for commercially supported software this means you must maintain a support agreement with the vendor for the lifetime of the application.	This will mean that if security issues are found you will be able to update the components your application is using. For open-source software this can sometimes be difficult to assess as maintenance is based on community engagement. If there is doubt, you should only use the component if you are capable of taking on its maintenance.
13	Security enforcing functionality must be defined in a minimal set of clearly defined and documented locations in the code. Document the location of security enforcing functionality	The impact of changes can be assessed more easily and it's easier to fix security weaknesses. By keeping security functionality in a small number of locations it is easier to understand and test than if it is spread across many locations in a large codebase.
14	Make sure current, tested versions of 3rd party components are used and maintained.	Older software versions can often contain bugs or security weaknesses which may lead to errors or unauthorised access to the application.
15	Access to the system must follow the controls in the Access Control & Management section below	This will make sure only authorised people have access to the application data

16	Default deployment of the system must result in minimum privileges.	This reduces the risk that a default deployment will result in un-needed high-privilege access. If this is needed, it should be explicitly enabled to prevent it being forgotten about during deployment
17	Software must execute with only the privileges it requires to perform its stated task. Where elevated privilege is required, it should be obtained in a well-known manner as documented in the security architecture, then relinquished as soon as possible	By running with minimum privilege the impact of security bugs or weaknesses can be reduced as exploiting them will only gain minimal access.
18	Tasks which require persistent elevated privileges must be separated from the main application software and subject to maximum scrutiny during peer review. Pay particular attention to tasks which call out /back to untrusted software.	Doing this means that any security bugs or weakness in the main application will not result in high-privilege access. By reducing the amount of code which runs at elevated privileges the risk of a security weakness or bug resulting in high-privilege access is reduced.
19	Create a plan for testing your application, including tests focused at testing any security controls or mitigations you have implemented.	
20	Define security boundaries which allow you to control: access rights to data authenticity & integrity of messages Validation of inputs and outputs	This makes sure controls are applied in the correct place to protect data in the application. Failure to implement controls in the right place may lead to unauthorised access to or modification of the data
21	Systems must ensure the integrity of data is maintained and preserved.	Poor control over the integrity of data could result in fraud, non-compliance with regulatory requirements such as Sarbanes-Oxley or industry standards such as PCI DSS.

3. System Delivery

Ref	Policy	Reason
22	Store your source code in an environment which restricts access so that only authorised people can make changes.	This will reduce the risk of unauthorised or malicious changes being made to the code which could compromise the application's security. Using a revision control system such as Git or SVN with access restricted to a group of named individuals is the easiest way to meet this control.
23	Ensure that all code changes are linked to a named individual. Each change must be auditable to identify who did what, when & how.	This means that any code changes (e.g. which introduces unwanted functionality) can be linked back to an individual. Using a revision control system such as Git or SVN with access restricted to a group of named individuals is the easiest way to meet this control.
24	Follow the best-practice guides relevant to your development environment or application type. Make sure you're aware of common issues for the platform or language you're using.	These are often published by the vendors of toolsets, or other third-party groups, Understanding and following them will help you avoid common issues:. A selection of commonly used guides is below: Web-Applications: the OWASP top 10 details common web-app vulnerabilities. Android Apps: CERT android secure coding standard: Windows / Windows Phone: iOS / Mac OS X: Apple's Secure Coding Guide: Linux: For more information or advice contact the standard owner.
25	Use a documented consistent coding	If coding styles are inconsistent across the application

	style	codebase it can become difficult to understand application logic and spot issues which may lead to unwanted behaviour. Most vendors produce a style guide for their language/environment, in the absence of any other requirement it is recommended that this is used. Some suggested style guides: Java – SEI CERT Oracle coding standard for Java C/C++ - CERT's secure coding guidelines or the MISRA-C guidelines C#: Python – PEP8 php – CodeIgniter's style guide
26	Use supported, trusted tool chains.	Supported tools are those that are readily available and actively maintained (either by a single vendor, e.g. Microsoft Visual Studio, or by a community, e.g. GNU Compilers Collection). Trusted tools are those provided (and signed) by a trusted party (e.g. Microsoft, Ubuntu) or are widely available and verifiable via well-known hashes (e.g. GCC source downloads). By using supported tools you will receive updates to fix bugs and other security issues. By verifying the trust of the tools you use you will ensure that your toolsets will not introduce any unwanted functionality.
27	Maintain tool chains.	Make sure all security updates are applied to build tools before use. This is to fix issues with the toolchain which could introduce unwanted behaviour in the output binaries
28	Use language and tool chain features that help identify or mitigate against simple	By enabling compiler or runtime warnings you can pre-emptively spot issues which may lead to security

	issues.	weaknesses or bugs (e.g. –fstack-protector in gcc will warn on potential buffer overflow conditions; or /GS or /SafeSEH in VisualC++ to make stack-based buffer overruns harder to exploit)
29	Use operating system features that mitigate against common attack vectors.	This is operating system specific, but you should enable security features which are available to add protection to your application, for example ASLR or DEP. Microsoft based operating systems support is available For linux, ASLR and DEP are usually enabled by default (for kernels after 2.6.12). SELinux is a way of enforcing access control policies and can be used to reduce the impact of an exploit. Redhat has a good guide on developing policies.
30	Sanitise all inputs according to the security architecture before processing. Convert inputs to a canonical form before santitisation.	If user-input is used as the basis for commands or database operations then the input can be used to cause unwanted operation e.g. SQL injection. Inputs can often take unusual forms, for example serialized java objects so it's important that you think about all cases where a user/intermediary might be able to modify or inject content. Inputs which are not converted can bypass sanitisation by using character sets the sanitisation was not designed to handle.
31	Sanitise all outputs to other systems according to security architecture.	Where user-input is used as output (either to another application or back to the user) then the input can be manipulated to cause unwanted behaviour in whatever is receiving the output (e.g. cross-site scripting)
32	Implement cryptography according to the controls in the cryptography section below.	Cryptography can be complex and difficult to get right. By following the appropriate controls you can be sure that the level of cryptography you are using is appropriate for the information you are protecting
33	Implement mechanisms to avoid unsafe	Some memory operations can result in an unprivileged

	memory operations. See OWASP	user being able to manipulate memory contents. In some contexts, this can lead to the manipulation of program flow (e.g. bypassing checks) or the execution of arbitrary code.
34	Use anti-tampering tools where indicated by the risk review	Allowing firmware or application code to be modified can bypass security functions. This can apply to mobile devices where you might want to stop your application from running on 'rooted' or 'jailbroken' devices. You may want to do this to make it harder for users to do runtime analysis/debugging to reverse engineer application functionality.
35	Use anti-reversing techniques where indicated by the risk review	Reverse engineering may reveal security functions. However, you need to remember that most anti-reversing techniques can only increase the time needed to reverse engineer, rather than prevent it altogether, so should not be the only form of security defence.
36	Avoid temporary files. Do not use tmpnam() or similar to generate file names.	Using temporary files can result in data distributed widely over a disk which can be recoverable if unauthorized access is gained to the system. tmpnam() should be avoided as because between generating the name and opening the file it is possible for another process to have created a file with the same name using tmpnam. This could lead to the other process being able to affect the integrity of the data stored, or read data it shouldn't have access to.
37	Do not use predefined (hard coded) passwords.	Passwords which are predefined in the application will usually become well known and shared among users. This can give unauthorised users access, or make it difficult to prove who had access as they are not assigned to a single user. It is also not possible to update and manage a hard-coded password in accordance with BT's specifications

		for account management.
38	Implement mechanisms to prevent unauthorised access to data in memory	Depending on your platform and risk review you may need to take steps to stop someone with access to the machine running your application from recovering sensitive data from memory. This may be by: • overwriting portions of memory once they are no longer needed • memory pinning • Using framework constructs designed to protect information, e.g. the SecureString class in .Net
39	Error reports must contain enough information to identify the cause of problems. Where errors are displayed to users however they must not give away information about application internals. For example, do not use generic exception types to deliver specific error conditions.	Incomplete error reports can frustrate investigations into issues, and hide unauthorised activity. An exploitation will often generate errors in the early stages while access it being gained, and if logged properly these can help the investigation into the issue. It is difficult to understand what caused the error condition if only a generic exception is logged. If the error condition was caused by an attack it is harder to deduce what operations were being carried out.
40	All software must be tested before migration into the live environment. Enact the test plan you created as part of the application design.	Untested software could: Introduce unacceptable security risks Not function according to requirements, especially security requirements detailed in the Security Registration Adversely affect existing operations
41	All security controls must be specifically tested to make sure they cannot be circumvented.	Unidentified vulnerabilities in security controls could be exploited in the live environment.
42	Test data must be deleted after a period	Disclosure of test data could provide an insight to the

	determined by the data owner.	donor system.

4. System Assurance

Ref	Control	Reason
43	Identified security vulnerabilities within the code or any components the code uses will be categorised using BT's vulnerability assessment criteria. Vulnerabilities must then be fixed in accordance with the timescale associated with each category. This may be adjusted based on the risk posture of the platform the application is within.	Un-fixed vulnerabilities can be used as part of an attack on the application or system.
44	Track and tag changes to fix security issues explicitly in change control	By tracking security fixes you will be able to quickly audit your application/environment and prove they have been applied
45	Install application releases which have been built from source using continuous integration.	Deployment from a clean build environment means that the application commences its operational life from a known set of code, and issues can be traced back to the source code which introduced them.
46	Make sure that the latest version of the application and any third party components it relies upon are used.	By updating third party components, the risk of a security issue in a component being exploited is reduced.
47	Apply patches to the environment hosting the application in accordance with the	It is important to apply security patches regularly as old software versions often become targets for exploitation.

	patching policy	A security issue in the hosting environment could leave application data at risk.
48	The following process areas must be defined and documented: Application of security patches Access Control Start-up/Shutdown Clock Synchronisation, Application and Job Management Data Transfers Monitoring and Alarms Problem and Escalation Management Back-up & Recovery Archiving Migration Change Control Disaster Recovery / Fallback Hardware & Software maintenance Log, review and action on unusual events	A lack of documented operating procedures often means that important processes get neglected, or the knowledge of how to carry them out leaves the team

5. Appendices:

5.1.Access Control

Ref	Policy	Reason

49	Define system access rights for users and other systems that interact with it. Access rights must be based upon: Operations to be performed by the system System to system interaction The User Access to Information and Systems policy Users must have the minimum privileges to do their job	Poor role definition could result in accidental or malicious operations being performed on the system
50	Shared Privileged* accounts must be managed as defined.	Without formal control it is difficult to track those responsible for security impacting changes.
51	Design application processes to run with the minimum access rights required for correct operation. Operations to be performed by the system System to system interaction Don’t use privileged accounts for application processes Document all privileged accesses Privilege escalation must use only known APIs Privilege escalation must be for the minimum required time only.	Application processes running with excessive privileges could be exploited to gain access to data or system privileges.

52	User sessions must be terminated after a maximum of 30 minutes inactivity. When time-out occurs the screen must be cleared of all displayed information.	Time-outs limit the opportunity for unauthorised access if the system is left unattended.
53	A user session must not exceed 12 hours.	To make sure users log on and re-authenticate at least every 12 hours.
54	Role and privilege-based access controls must be provided on all management ports, whether local (Craft/ console terminal, or terminal server) or remote (via EMS) ACLs must be applied on all (IP) management interfaces, restricting the source address and port, and protocol invoked; in particular, the ability to restrict access to ICMP, HTTP, SNMP, FTP, TFTP, SSH, Telnet and advanced routing protocols such as OSPF.	Exposed management interfaces can be exploited for unauthorised access.
55	Only securely authenticated management protocols must be used e.g. SNMP v3 (AuthnNoPriv as a minimum) SSHv2 - see cryptography section HTTPS - see cryptography section.	Insecure management protocols can be exploited for unauthorised access.

5.2 Cryptography

Ref	Policy	Reason
56	Current Cryptographic libraries must be used.	Cryptographic libraries are updated regularly. In addition to updating software packages in-line with vendor direction cryptographic packages should be reviewed and updated regularly.
57	Only use approved industry standard cipher suites for encryption. E.g. for TLS SSLv2. Note the warning above. If in doubt get advice from ITSAC.	Non approved ciphers may introduce vulnerabilities.
58	The latest version of TLS must be used for new deployments. SSL V1,2 & 3 must not be used.	Earlier versions, up to and including TLS1.0 are no longer considered secure.
59	Perfect Forward Secrecy must be enabled.	Perfect forward secrecy algorithms prevent captured messages being decrypted even if the authentication private key is compromised in the future.
60	Self-signed certificates must not be used.	Self-signed certificates negate the benefit of end-point authentication and also significantly decrease the ability for an individual to detect a man-in-the-middle attack.
61	Passwords must be protected using a non-reversible one way mathematical function (e.g. Hashing algorithm) with a unique randomising factor (Salt) per password.	Stored password files can be extracted and as such all entries must be protected to prevent recovery of cleartext passwords.
62	Protected passwords must be stored away from a system's configuration files and have access control implemented so that only appropriate privileged users can	It must never be possible to retrieve protected passwords by directory traversal, SNMP walk, configuration dump, or other mechanism, which might allow attempts at offline cracking.

	read or copy the contents.	

5.2.1 Web Cryptography Controls

Ref	Policy	Reason
63	Cookies which store or are used to access In Confidence or higher level data must be transported securely.	Cookies can be stolen through several methods such as XSS and sniffing.
64	Secure and non-secure content must not be mixed on the same page.	Non secure could potentially steal secure info from the content.
65	TLS must be used for all login pages and all authenticated pages. The login page and all subsequent authenticated pages must be exclusively accessed over TLS. The login page and all subsequent authenticated pages must be exclusively accessed over TLS.	Failure to use TLS for the login landing page allows an attacker to modify the login form action, causing the user's credentials to be posted to an arbitrary location. Failure to use TLS for authenticated pages after the login enables an attacker to view the unencrypted session ID and compromise the user's authenticated session. Failure to use TLS can result in a man-in-the-middle attack
66	Clients must be instructed not to cache sensitive data.	The TLS protocol provides confidentiality only for data in transit but it doesn't help with potential data leakage issues at the client.
67	Use nationally and internationally accepted digital signature standards	The use of non-standard digital signature techniques could result in misplaced trust in the signature; national and international regulation places controls on digital summarising the import, export, and domestic crypto controls around the world.

68	Wildcard certificates must not be used - under any circumstances.	Wildcard certificates are a more attractive target. They can certify an unintended server and make an encryption domain vulnerable. If one server or sub-domain is compromised, all sub-domains may be compromised. If the wildcard certificate needs to be revoked, all sub-domains will need a new certificate.

5.2.2 General Key Management

Ref	Policy	Reason
69	Cryptographic keys for all new deployments must meet or exceed minimum standards.	Short or weak keys can be easily compromised within the lifetime of the data.
70	Symmetric & Asymmetric Keys must be generated using approved tools and libraries	Unapproved tools and libraries have the potential to generate weak keys.
71	Passwords controlling the use of cryptographic keys must provide protection that is equivalent to the key itself.	Weak passwords negate the strength of the cryptographic key.
72	A senior manager within BT must be responsible for the security of the key material.	They will have the seniority, capability and trustworthiness that is commensurate with the security of the data that the keys will protect.
73	The senior Key Manager must make sure the responsibilities in the detail column are allocated and carried out by suitably	Responsibilities Key Management policy

	qualified and trained personnel:	Key generation or acquisition Design of the key distribution, key periods, revocation and accounting structure Creation of key management procedure Operation of the key management Protection of private keys and related materials Emergency procedures, such as revocation Auditing of key operations Key recovery Destruction of keys

Document Control

Third Party-Industry Guidance Standards on 'Secure Coding'

Author: BT Security

Issue 1, published October 2016