Cyber-Security Developer Checklist

This checklist is designed to be a handy reference for developers creating software for external (public) use or internally within an organisation.

Development environment *This is the programming language or environment you use to develop your software – for example, Visual Studio, Microsoft Access, Alpha Anywhere, Python...* *The principles apply no matter what you use to build software*	❑ Are you using the latest version (or a supported version) with all patches applied? *If you are using older versions of the language or environment, there may be unpatched security vulnerabilities which could result in insecure software being built.* ❑ Have you reviewed any documentation associated with the environment which is specifically related to security? *Most languages and environments have guides to help developers to build software in a secure way. Make the most of them!* ❑ Do you subscribe to any news feeds or blogs from the developers of the environment so that you are alerted to any known vulnerabilities? *When vulnerabilities are found, patches are often made available within days. Make sure you know how you would be told about them and apply any updates as they become available*
Coding standards/methods *If you follow basic good practice when developing your software, it will be easier to develop in a way that focuses on security*	❑ Is version control used so there is traceability over what changes have been made to the software, when were they made and by whom? *Examples of version control products are Git and Subversion – if you don't work with version control software you are missing a trick!* ❑ Can you use your version control software to easily revert changes if necessary (such as an update going wrong and the software needing to be restored to a previous working version quickly)? *That's one of the major benefits of version control systems – and very useful if you find you have deployed any sort of bug, especially one that relates to a security vulnerability* ❑ Have you ensured that nothing sensitive is committed to version control? *This includes things such as API keys, especially important if you are working with a public repository where hackers may be looking at your code to find areas to exploit*

	❑ Are you ensuring that all user input is sanitized? *An example would be parametrizing all SQL commands that use user input to prevent SQL injection attacks. SQL injection is one of the most common vulnerabilities used by hackers to gain access or disrupt a system* ❑ Are you coding to your own or an agreed set of standards? *If you are working in more than one language, there may be multiple sets of standards. Working in an agreed, consistent way help to avoid the introduction of all types of bugs. If you are not working to standards, you may need to consider a risk assessment to judge whether the approach is a valid one* ❑ Are you ensuring that all third-party libraries being used are kept up to date? *Software often relies on open-source or commercially licenced libraries which add functionality to a system. You need to be careful that these libraries are kept up-to-date and patched with any security updated provided by the supplier/developer of the library* ❑ Is there a process of code review so that two pairs of eyes have checked the critical code blocks and methods (especially where user access or other security code is concerned) *An example would be requesting reviews when performing pull requests in GitHub*
Databases (if any) *Are you following best practice with regards to any databases your system uses? Examples of databases technology are Microsoft SQL Server, MySQL, MongoDB*	❑ Are the databases hosted on an inherently secure platform? *Examples would be Amazon AWS or Microsoft Azure, or a database server hosted in a professional datacentre with appropriate certifications* ❑ Are development, test and production database in place? *It is a bad idea – for lots of reasons, not least those relating to security – to muddle the purpose of a database* ❑ Is the database itself encrypted? *Most database technologies allow you to encrypt the entire database "at rest" meaning that all data is stored in an encrypted form and hence has limited or no value even if the server itself is compromised*

	❑ Is the connection from your software to the database encrypted? *Most database technologies enable the connection from the client to the server to be encrypted preventing a hacker from "eavesdropping" the connection and intercepting data.* ❑ Do you/your database administrator understand encryption and how crypto keys should be managed? *If not, it is worth researching or employing an expert to help* ❑ Is there an automated backup process in place with rehearsed data recovery tests? *It is vital that any database has automated, regular backups and those backups must be regularly tested to ensure that the process is working correctly. Think hard about where the backups are stored to ensure that they are physically separate from the source data* ❑ Is the database server/firewall configured to only allow connections from essential locations? *This may not be possible (depending on the application), but it may be possible to restrict access to certain locations (IP addresses) only – or to only allow access within a VPN. If this is not possible in general, admin (superuser) access to the database should be restricted to necessary locations only* ❑ Have appropriate accounts been created so that users accessing the database via your application only have the minimal access rights that they need? *For example, you should never code an application to use an admin/superuser account*
User access controls *How do your users authenticate themselves with your application? Is the method secure?*	❑ Is the user table encrypted or stored within an encrypted database? *If not, why not! Think about the consequences if the user table was hacked* ❑ Are passwords stored in plain text or are they hashed using a secure algorithm such as SHA-256? *Never store passwords in plain text! Just consider what would happen if the user list was accessed either accidentally or through discovery by a hacker* ❑ Is it possible for anyone (including you) to see a user's password? *This should not be possible – users should need to reset a password if they forget it and their*

	password should not be visible or accessible to anyone. If passwords are accessible to you, as developer, they are theoretically available to a hacker, too ❑ How are initial account passwords delivered and are account holders forced to change the password on initial access? *Get this right from the start – if you launch the system with a less-than-ideal password reset mechanism it will be harder to fix and it may get forgotten* ❑ Have you included a mechanism for users to reset a lost password or to issue a new password? *None of this should require you, the developer, to get involved – these processes should be built into your software* ❑ Have you included password rules including length and complexity? *Some organisations may also include other password management rules such as duration, restricted login times and locations. You should also consider multi-factor authentication which is becoming a standard, now* ❑ Although inconvenient to users, consider disabling the option for users (or browser) to cache passwords *If you* do *allow your users to store their credentials, make sure they tick a box to say that they are happy for the system (or browser) to retain the information – make it their decision, not yours!* ❑ Are users forced to update/change their passwords on a regular basis? *Also, is there a way to force all users to choose a new password if a compromise is suspected?* ❑ Is your system coded to handle a "brute force" password attack? Do users get locked out after a specific number of unsuccessful attempts to log in? *Brute-force attacks are commonplace on publicly-accessible systems. Don't think of this scenario as "if" but "when" as it is sure to happen to you*
Audit logs	❑ Are all logins (successful and unsuccessful) logged and capable of review? *This is an easy feature to deliver in your software and is often invaluable if an incident takes place*

Audit logs are important in any system. They provide means to review what happened if there is a security breach or one is suspected	❑ If there are multiple failed login attempts, is a system superuser automatically notified? *You should also lock the user out for a period to protect them and the system* ❑ Are critical data changes logged to a file? *For example, if a user changes their password, is this change (not the password, of course!) logged to an audit trail so it can be reviewed?* ❑ Are changes to the database schema and software/product recorded against the account that initiated the activity? *Again, most database products can provide this level of logging either built-in or via the use of third-party utilities/scripts* ❑ Have you built an option into the system to record *all* data changes if the users require it? *This can be useful if there is a worry that a system has been hacked or affected by a virus and data has been changed on an ongoing basis*
Security testing *Even if you are building your system in a way that minimises security risks, it is important to test your application for vulnerabilities, too*	❑ If your application is public facing (for example, it is an application available on the Internet where customers can sign up and use the system without intervention), have you had the application penetration tested? *This isn't a one-off activity - you should have plans to regularly pen-test the application*
Production environment (for a web app) *This is the environment in which the software is hosted. This could be a physical server, a cloud server, serverless model or could be hosted in a managed environment* *This checklist assumes that you are developing a web app – if you are developing a desktop app or a mobile app, these may not apply*	❑ Is the web server using the latest version (or a supported version) with all patches applied? *This is essential – hackers look for out-of-date web server software to exploit using known vulnerabilities* ❑ Is traffic between the client and server encrypted using TLS (v1.2 or greater)? *HTTPS is a fundamental, now, and unencrypted sites (HTTP) should never be used* ❑ Is the hosted environment secure? *The server hosting the software (if not using serverless or managed hosting approaches) should be protected with a secure password & 2FA and should be restricted so that only certain IP*

	addresses can remote onto it by setting rules on the firewall ❑ Is source and compiled code separated? *Don't mix source code and development environments with production servers – only published/compiled code should be deployed to a production server* ❑ Are there any open ports on the server that do not need to be open? *Port scanning services can be easily used to check this what ports are accessible to the internet* ❑ Is the OS kept up to date and are security patches installed regularly? *Again, hackers look for servers running obsolete versions of (eg) Windows Server*
Before you go live *There are important checks to make before your system goes live*	❑ Have all test user accounts been disabled? *You may have issued test accounts while you were testing the system – these should be disabled before the system goes live* ❑ Is there a documented method to disable the system if there was a security breach or DDoS attack? *You don't want to be "making this up" in the event of a system emergency! Create your procedure, rehearse it, and make sure that the location of the documentation is shared in case you are not available when disaster strikes* ❑ Is there adequate user and technical documentation? *No-one likes to write documentation and there is a suspicion that no user will read it anyway! However, professional documentation, for users and administrators, is essential for the key (security) processes in your system*

Robin Bennett (with additions by Tom Watkins tom.watkins@start-software.com and incorporating suggestions from Chris Myers chrismyers@ntlworld.com chrismyers@ntlworld.com)
robin.bennett@start-software.com

Draft 1: 14.12.2021
Draft 2: 21.2.2022