# Design patterns for secure software development: demonstrating security through the MVC pattern

MR Colesky
Nelson Mandela Metropolitan University
Port Elizabeth, South Africa
Michael.Colesky@nmmu.ac.za

LA Futcher
Nelson Mandela Metropolitan University
Port Elizabeth, South Africa
Lynn.Futcher@nmmu.ac.za

JF Van Niekerk
Nelson Mandela Metropolitan University
Port Elizabeth, South Africa
Johan.VanNiekerk@nmmu.ac.za

***With the current interconnected state of our world, the security of every system is important. More often than not, however, security is treated independent to software. This overlooks a vital security principle; security should be grounded in the same early design and development stages as an application itself. By instead favouring immediate returns over long term benefits, the value of concrete security is commonly undermined. To remedy this, this paper seeks to explore the relaying of security into a fundamental building block of software; namely design patterns.***

***Keywords: Design patterns, information security, security principles, secure software, development software***

## 1. Introduction

A well-argued point in information security is its need for human education, training and awareness. Quite often, it is the human aspect which is the weakest link. While this factor cannot be stressed enough, the technical factors should not be ignored. It does not help if users are well trained to make use of applications securely if the application itself is not designed with security in mind (Hight 2005:1). If the underlying code is not inherently secured then the application's overall ability to resist security attacks falls flat.

The introduction of the .Net Framework, Python, Java and other Runtime Environments, however, has been able to reduce some security risks. Much of what these managed frameworks reduce is the need for repetitive tasks such as checks and exceptions (Howard & LeBlanc 2009:540–556). However, this does not guarantee secure coding.

A single overlooked weakness can be devastating. Without incorporating security at every point in an application's lifecycle, the potential for oversight is substantial (Breu, Burger, Hafner & Popp 2004:1). Without ensuring that it is applied at the root, the coating of security will not effectively integrate with the software.

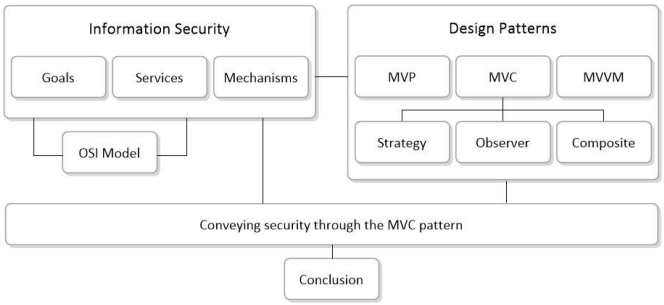
This falls back to the original statement. There is a need for human training, education and awareness; however, this applies to developers as well. Developers need to be aware of how easily an insecure application can be exploited, and how heavily such an exploit affects everything around it. Developers therefore need proper training in being able to secure their software (Stoneburner, Hayden & Feringa 2001:17). This is sometimes catered for, though often programmers gain their qualifications without any practical knowledge of secure software development.

One effective tool in conveying good design principles is the use of software design patterns (Freeman, Robson, Bates & Sierra 2004:9). In the study and practical application of these development blueprints, students begin to grasp not only the concepts of re-usable design, but also of good coding practice.

Effectively, the main aim is to make secure coding as natural as coding itself. If this can be done, then these ideas of how to design secure solutions will permeate throughout industry. Rather than having to incorporate security as a separate and time-consuming after-thought, security can be integrated in the same early design stages as the code itself.

This paper examines how design patterns can play a role in introducing security to the groundwork of applications by influencing coding habits of developers. Figure 1 shows the structure that this paper will follow.

**Figure 1: Paper outline**




First to be addressed is information security. This will funnel from the general into the specific, using network layering to guide from goals into lower level mechanisms. A discussion of design patterns follows. These are then combined in the relaying of secure concepts through patterns; the precursor to the conclusion.

## 2. Information security

Security is like a well-trained pugilist on defence. Adapted from Khan and Mustafa (2008:1622), 'secure software' provides solutions which block the vast majority of attacks, withstand most of what it cannot block, and recover quickly with minimal damage.

This notion of security being no impenetrable wall is one also acknowledged by Bishop and Engle (2006:15). In this, there is no silver bullet. A best effort must be made to cover all the bases. Breu et al (2004:1) suggest that this effort be handled in every stage of the development life cycle. This tactic is again given mention by Jones and Rastogi (2004:1).

In particular, emphasis is made on security being a specific requirement. However, it has been identified that in industry, security is often seen as a diversion from productivity (Jürjens 2002:2; Khan & Mustafa 2008:1622).

Companies are all too eager to cut development time and costs, and quite often good security is ignored when not explicitly requested.

Instead of designing code securely, structurally from the inside out, programmers try to rely on instinct. This aspiration to successfully implement security as a habit is one commonly left unsatisfied. All too often developers are not given the training to use such intuition (Khan & Mustafa 2008:1622).

Furthermore, in a slideshow presentation given by Black Hat's Drew Miller (2003) at a Windows security conference, it is shown how the introduction of managed code in the .Net Framework changes the landscape for application security. Notably, the risks of buffer overflows, code and role access security, as well as the Trojan-horse style virus are found to have been mitigated to a certain degree. Further examples of these include memory corruption and overflows in general, as mentioned in Mano Paul's "The Ten Best Practices for Secure Software Development" whitepaper from the book: Official $(ISC)^2$ Guide to the CSSLP (2011).

To elaborate on this, many exploits are more difficult to produce when code is in a lowered trust state. With managed code, buffer overflows would have to be exploiting a weakness of the managed framework rather than a weakness of the application. It is when unsafe and unchecked constructs are used that the issue becomes a noticeable threat. In properly managed code, the introduction of self-resizing variables makes overflows less of a concern (Drew Miller 2003:5).

With the aforementioned lack of training, as well as with the popularity of managed coding languages already mitigating a few of the risks found traditionally, developers are less likely to pay specific attention to security. The following section begins with information security goals.

### 2.1. Information security goals

In information security, controls are implemented to protect the confidentiality, integrity and availability (CIA) requirements of a system and its information (National Institute Of Standards and Technology 2011:1). In the business world, this means creating an information security programme, which leads to the identification of risks, and the mitigation of those risks through technical controls and policies. These need to be re-evaluated continually. However, in the context of a software solution, a different approach is taken.

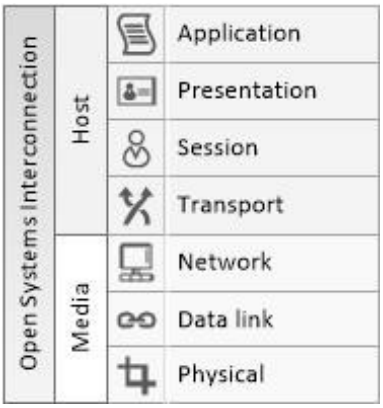
As previously mentioned, software development does not always cater for good security. When it does, however, according to Polydys, Ryan and Ryan (2006:56), it should address security at every stage of development. This is essential in maintaining an image of trustworthiness, and without security 'baked in', it is 'difficult or impossible' to provide a secure solution.

Polydys et al (2006:56) also mention predictability and conformance. When considering the information security goals we can map these to one another. A client needs to trust that their data will be secured; the software needs to follow strict information integrity policies to ensure that it is not modified without the required permissions and authorisation; and functionality should work whenever needed without misbehaving or disappearing. These goals are addressed through the information security services found in the Open Systems Interconnection (OSI) Reference Model as discussed in the following section.

### 2.2. The OSI model

The OSI model describes the communication between abstracted 'layers' of a system in communication networks. These layers can represent progressively higher-level forms of data as one goes up each layer as shown in Figure 2.

**Figure 2: The OSI model (adapted from Miller 2003)**




The bottom-most layer describes the physical communication media, signal and binary transmissions. The data link layer handles physical addressing, where above it, the network layer handles logical and port addressing. These make up the media section of the OSI model.

The transport layer handles connections as a whole, while the communicative session layer is commonly handled by web sites and services. The presentation and application layers are the most addressed in code. These handle actual data objects and functionality respectively. This is the host section of the OSI model.

As noted by Rachelle L. Miller (2003:7), the OSI model is not designed to enforce a structure. This paper's main focus is on the development related aspects derived from the security services of this model. These security services are discussed in section 2.3. together with security mechanisms in section 2.4.

### 2.3. Information security services

In securing Information Systems (IS), there exists a well-used framework which addresses how software should serve an organisation's security as well as the means to do so. The security services of the OSI Reference Model in ISO/IEC 7498-2 (1989) are often provided as a base for analysis of software security. These are:

1. Identification & Authentication
2. Access Control
3. Data Confidentiality
4. Data Integrity
5. Non-repudiation

These services are provided with greater depth. Firstly, identification & authentication refers to both data origin (that the peer is the source of the data) as well as the peer entity authentication (that an entity is who it claims to be).

Secondly, access control manages what resources are allowed to be allocated. Resources refer to not only the ability to read and write to devices, but also to execute a process or service. Hight (2005:2) mentions that a system's access control is largely affected by how well users play their part in security.

Thirdly, in data confidentiality, the protection of data from unauthorised disclosure, specific lower level services are provided (International Telecommunication Union 1991:11). There is firstly, the confidentiality of the traffic flow itself. If this is left open to inspection, conclusions may be derived from it. There is then the confidentiality of an entire connection, or of a single Service Data Unit (SDU). SDU refers to a single instance of sent data which has not yet been encapsulated. Within either of these, there lies selective field confidentiality – providing protection to only the specific fields that require it.

Fourthly, data integrity refers to the assurance that the data is unchanged. This can take place in the same full connection, connectionless and selective field options as data confidentiality. Data integrity may be checked for a full connection with or without the ability to recover from detected disparities. It may check for a 'connectionless' single SDU. One may elect to only check selective fields in either option.

Finally, there is non-repudiation. This refers to the inability to deny that a transaction took place (Alkussayer & Allen 2010:98). The service records transactions so that there is evidence of everything which transpires. This can take place in either or both of the recipient and sender receiving transactional evidence.

### 2.4. Information security mechanisms

As described by ISO/IEC 7498-2 (1989) and X.800 (International Telecommunication Union 1991:10–12), the following eight mechanisms may be utilised to achieve the security services:

1. Encipherment
2. Digital Signatures
3. Access Control
4. Data Integrity
5. Authentication Exchange
6. Traffic Padding
7. Routing Control
8. Notarization

Encipherment, or encryption, refers to the rendering of all data to be illegible. When reversible, encipherment may make use of either symmetric or asymmetric encryption. Irreversible encipherment however, need not make use of either if desired; this type of encryption can make use of hash functions to irreversibly transform information. In symmetric encryption, a single secret key may encrypt data as well as decrypt that same data. Asymmetric encryption, on the other hand, requires two keys: one to encrypt, and one to decrypt. The use of reversible encryption requires strong key management (Swanson 1996:51).

Digital signatures are identifiers attached to messages to uniquely differentiate between peers. This includes both the signing and verification of signatures. Encryption may also be used to facilitate signatures (Swanson 1996:57). Like asymmetric encryption, this makes use of both private and public keys. Verification makes use of the public key to deduce whether the message has been manipulated with the private key. Because the private key is only held by the sender, verification of the public key proves who sent the message.

Access control mechanisms refer specifically to the actual allowing and disallowing of access to resources. The relation can be understood by comparing access control to a private establishment's bouncer. If you do not have access rights, you will not be granted access. This metaphor can go further in describing access control lists, pre-authenticated passwords, status of the entity, time/route of access and duration without much elaboration.

Data integrity mechanisms check for equality between data before and after retrieval. This can be at block or stream level. Data integrity can be ensured through the use of checkvalues or block checks which may be encrypted to ensure no modification has been made. If a modification is identified, this may include automatic recovery, error-correction or retransmission.

The authentication process is handled by authentication exchange mechanisms. This utilises typical password usage, but can also make use of encryption and credentials as authentication indicators (Swanson 1996:43–45). Authentication may also involve other mechanisms such as with signatures and notaries. Of particular interest in this regard is that of unilateral (two-way) and mutual (three-way) authentication through handshakes. This can be used for encryption to protect against replaying an encrypted transmission.

Combined with encipherment or a similar confidentiality service, traffic padding mechanisms are used to obscure the patterns of network traffic. This prevents external analysis from deriving important weaknesses in the network. According to Jiang, Vaidya and Zhao (2003:1–2), traffic packets are spoofed to disrupt any analysis – this is referred to as a *cover mode*. In doing this, potential attackers come to false conclusions.

As indicated by Yi, Naldurg and Kravets (2002:1), routing controls may be implemented to bypass less trusted nodes in a network when secure information is concerned. The ability to dynamically, or through prearranged metrics, determine the most secure route through feedback is the basis of the routing control mechanism (International Telecommunication Union 1991:12).

As described by Nakahara (2000:1), "[a notarization] safely stores the evidence of events and actions during trading for a long term and subsequently certifies the fact by presenting formatted evidence (notary token)". The X.800 states much the same, adding that this evidence includes integrity, origin, time and destination. The notary can be viewed as a third party that is trusted by both persons, handling all transferred information. This communication can also make use of other non-repudiation mechanisms as well as encipherment.

So far this paper has examined security from the top down. In the following section, approaches to software design are described in the form of software design patterns.

## 3. Design patterns

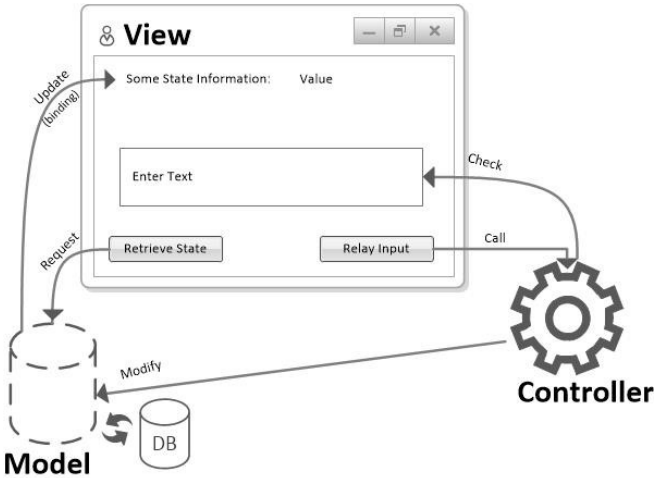
In software development, software design patterns are used to provide a guideline for the resolution of commonly occurring problems. These patterns typically make use of *class diagrams* or similar visual representations which represent the core of the generalised solution. This is the method of demonstration in use for this section, which focuses on areas around the Model-View-Controller (MVC) pattern.

### 3.1. Model-View-Controller

The MVC pattern focuses on separating application design into common developer roles: back-end components, User Interface (UI) design and interfacing with functionality between these (Smith 2009:2).

The view shows the windows, buttons, and other controls to the user; the controller interprets clicks and other commands; and the model does the business logic and object retrieval – then relaying the changes to the view again (Freeman et al. 2004:536–540). The view can also request state information from the model, and the controller can ask the view to update its display. This relationship is shown in Figure 3.

**Figure 3: The Model-View-Controller (adapted from Freeman et al. 2004)**



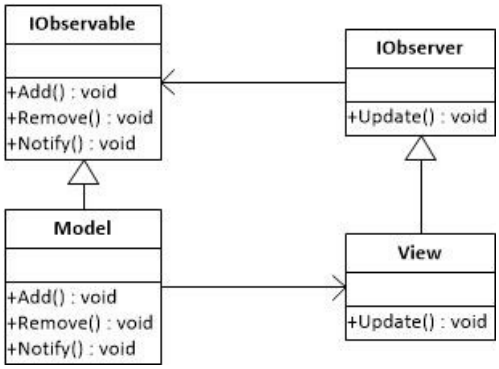

The Model-View-Controller features concepts found in other patterns (Freeman et al. 2004:541). These concepts are briefly explained below.

### 3.2. Observer

The view is an observer of the model. It registers to be updated when state changes. This registration is shown in Figure 3 on the button entitled 'Retrieve State'. However, in reality, this happens behind the scenes. Using Figure 4, this can be understood in the following manner.

The view can access the model to add itself as an observer. When something changes, notify calls for all observers, in this case the view, to update their state. Observer has been identified as one of the most useful patterns (Zhang & Budgen 2012:7).

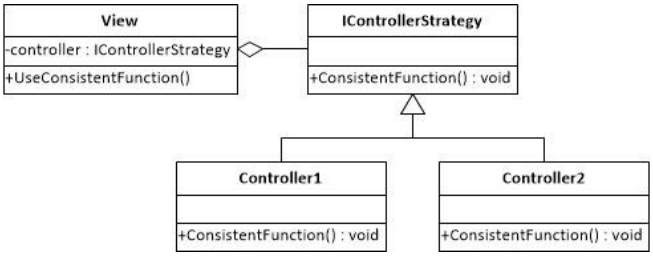
**Figure 4: MVC perspective observer (adapted from Freeman et al. 2004)**




### 3.3. Strategy

The view is also able to make use of a selection of controllers which implement its required functions. The ability to swop out these controllers works in the same way as the strategy pattern (Figure 5).

**Figure 5: MVC perspective strategy (adapted from Freeman et al. 2004)**

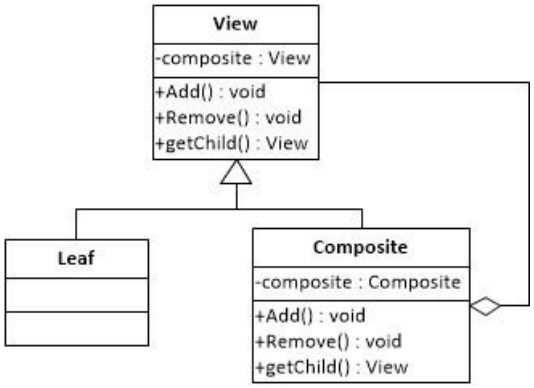



This pattern describes keeping the structure of components consistent so that they can be used interchangeably (Freeman et al. 2004:24). The view is effectively able to use any controller without having to worry about specific implementation.

### 3.4. Composite

Finally the view is also able to compose of further views. A control can comprise of a set of other controls and each of their functions will be handled as if only one control existed. The ability to contain multiple objects and act interchangeably as one object or many is shown in the composite pattern (Figure 6).

**Figure 6: MVC perspective composite (adapted from Freeman et al. 2004)**




A view can compose of any number of components. Composite components can do the same. Leaf components do not. An understanding of recursion is needed to really grasp this pattern; however, some consider this pattern crucial (Zhang & Budgen 2012:12).

### 3.5. Model-View-Presenter

The MVC is an age-old pattern which has been revisited many times (Holmström 2011:18; Smith 2009:2). First the Model-View-Presenter (MVP) modified the concept of a controller to a presenter. The presenter handles interaction between the model and view; updating the view and reacting to user interaction.

The MVP splits into two flavours; the passive view where all view logic must be separately coded; and the supervising controller where the view is bound to the model via data-binding and the controller handles user interaction. The disadvantage of the supervising controller is the coupling between view and model – which is less favourable for testing.

### 3.6. Model-View-ViewModel

Abstracting the view of the MVC, the presentation model was the basis for the Model-View-ViewModel (MVVM). The MVVM directly binds view and viewmodel. The viewmodel encapsulates all view behaviour and this allows for potentially no controller. As such, instead of saying that the MVVM is a specialised version of the MVC, it is more correct to say that it is a specialisation of the presentation model towards Windows Presentation Foundation (WPF) (Smith 2009:2).

The MVVM is rarely used within web applications as it is tailored for WPF. Similarly, MVP is tailored to forms. However, the standard MVC is widely used in web development, especially for ASP.NET. As such the pattern selected to relay security concepts is the MVC pattern.

## 4. Conveying security through the MVC pattern

As stated in the beginning of this paper, security mechanisms will be used to structure the delivery of security concepts. The MVC pattern will convey them. The mechanisms most relevant are access control, authentication exchange and encipherment. However, for web development, notarization, data integrity and digital signage are also relevant. The remaining two are constrained by specific network applications.

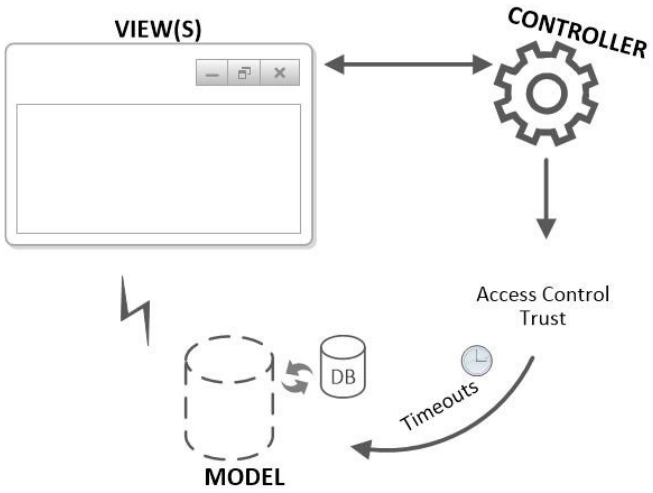
### 4.1. Access control

Access rights, privileges and trust are a very important aspect of software security. A substantial amount of the enhanced security from managed frameworks, specifically the .NET Framework, is based off of trust (Howard & LeBlanc 2009:536).

This embedded security makes the developer's life easier. Though, this does not mean that a programmer should ignore security concepts which are handled by their choice of runtime, as its contributions are invalidated if the coder does not use them properly (Howard & LeBlanc 2009:535).

The principle of 'least privilege', for one, is not only to provide an entity with as little rights as possible, but also requires that access be permitted for the shortest duration possible. This aspect is depicted in Figure 7.

**Figure 7: Access control using MVC (adapted from Freeman et al. 2004)**




This goes further in allowing deliberate restriction and demanding of access when an application is strong-named. The idea of limiting privileges applies especially to web development because web functionality is based on partial trust. The .NET Framework restricts these calls by default, which protects against many online attacks (Howard & LeBlanc 2009:540–556).

### 4.2. Authentication exchange

The fifth most overlooked vulnerability according to the Common Weakness Enumeration (CWE) top 25 (Martin, Brown, Paller, Kirby & Christey 2011:3) is the 'missing authentication for [a] critical function'. Following closely are 'missing authorization' and 'hard-coded credentials' at sixth and seventh place respectively.

The first refers to only guarding the 'front door' (Martin et al. 2011:14). As with access control, assess what areas require what role authentication for access. Otherwise, deny by default. When designing authentication measures, stick to using known solutions provided by the framework, and authenticate on the server rather than the client. For automatically denying access, turn to application firewalls.

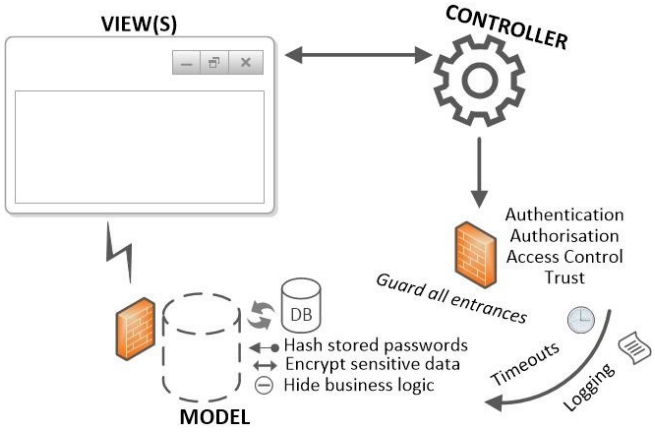
The caller must be able to provide evidence that it is who or what it says it is (Howard & LeBlanc 2009:109–118). Even so, its access may only be granted with certain levels of trust. Every request should be recorded, attempts should be restricted, sessions should timeout, and this should happen in a manner which does not permit abuse.

Though less severe than missing authentication, storing hard-coded access is forbidden; it provides another back door for the determined hacker. These aspects can be seen along with encipherment concepts in Figure 8.

### 4.3. Encipherment

The CWE top 25 lists 'missing encryption of sensitive data' as the eighth most common weakness (Martin et al. 2011:3). At nineteenth position is the use of a 'broken or risky' algorithm. Similarly, in Howard and LeBlanc's Writing Secure Code (2009:259–269), mention is given to the unintentional use of a predictable random generator for encryption. In .NET for one, the Cryptography namespace should be used rather than any random function.

**Figure 8: Access control, authentication and encipherment using MVC (adapted from Freeman et al. 2004)**




Data held within the model should be kept secure. Where it is identified to be sensitive, the data should be encrypted with tried and tested algorithms. In the case of passwords, however, there is no need to have a reversible cipher. Store the hash of the password and then hash whatever entry requests comparison as their values will be the same. This and all other business logic should be kept confidential as well. These aspects are also reflected in Figure 8.

### 4.4. Notarization, data integrity and digital signage

While the remaining mechanisms can be approached separately, they do not all correlate to individual principles. Integrity is enforced by comparing checksums or hashes before and after transfer, though in terms of poor software security, this mechanism can also cover injections and overflows. Notarization and signage fall under non-repudiation, which is less relevant within the strict development context.

Non-repudiation is not something to ignore, however, and it should be accounted for in functions which specifically require it. As mentioned by Stoneburner, Hayden and Feringa (2001:24), securing an application does not require the use of all mechanisms.

### 4.5. 'All input is evil'

As noted by Howard and LeBlanc (2009:341), "All input is evil". The top four weaknesses defined by Bob Martin et al (2011:3) are SQL (Structured Query Language) injection, operating system command injection, buffer overflows and cross-site scripting respectively. These all correlate to malicious input.

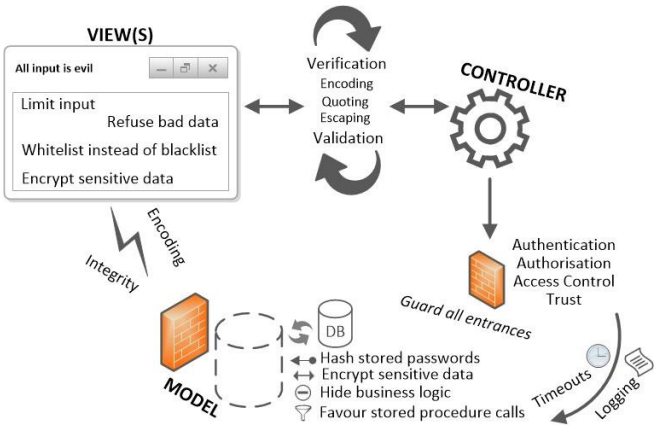
These threats can, however, be reduced (Martin et al. 2011:3–14). First and foremost, suspect all input as being malicious; favour drop-down lists over combo boxes; and reduce any and all freedom for expression where possible, rather whitelisting than blacklisting.

This includes escaping, encoding and quoting characters properly, using validation, and cleansing input from all sources, even hidden fields, cookies, and URIs (Uniform Resource Identifier). These tactics are commonly utilised by browser and website server software. This includes *HttpOnly* session cookies for Internet Explorer and FireFox as well as the *filter* option for Apache Struts (Martin et al. 2011:13).

In the interest of further protection from injection, make sure that all calls to the database are either stored procedures, previously prepared statements, or assigned only by parameter. Ensure that retrieved data is not modified during transmission. This is done by creating checkvalues which uniquely identify the data. Note that checkvalues should be encrypted, though, since a checkvalue may be recalculated.

The interaction between the view and controller needs to be closely checked against malicious input. This is best handled by limiting the scope for unexpected submissions. In validating and again ensuring that received information conforms to strict encoding, the controller can protect against these threats. The same applies to client side code. The controller should not assume that all script is legitimate.

**Figure 9: Conveying security through MVC (adapted from Freeman et al. 2004)**




Similarly, when the controller communicates to the model, rights to access should not be assumed. In all points of access there must be stringent procedures to verify that whatever calls a server-side business logic function has the authorisation, credentials and rights to do so.

Figure 9 depicts the correlation between some key security aspects and each component of the Model-View-Controller. The model shows how a pattern's representation can encourage emphasis on security in certain areas of an MVC web application.

In general, the use of structured frameworks which protect against these vulnerabilities is recommended. In addition, every aspect adding to the difficulty to exploit a weakness is helpful.

## 5. Conclusion

This paper set out to determine how software design patterns can be utilised to convey security concepts and principles. In designing a model reminiscent of the original pattern's depiction, with additional software security placed strategically, the components of a secure design may be observed.

This abstraction should allow those familiar with the pattern to grasp the security ideas embedded without the significant investment needed in reviewing these separately. The collaboration of these elements into one representation is useful in reviewing these concepts even if they are already known.

Akin to design patterns, these are there in simplified format for reference when needed; either conceptually or when in need of a design solution. Similarly, this illustration is by no means prescriptive. It is a generalisation to be consulted as a frame of reference; to be moulded into whichever form is most useful.

Particularly, individuals making use of design patterns in this way should be able to consider security from the beginning. As such, it is encouraged that further concepts be relayed through design patterns, especially with regard to security.

## 6. References


Alkussayer, A. and Allen, W.H. 2010. The ISDF Framework: Towards Secure Software Development. *Journal of Information Processing Systems* 6(1):91–106. Available http://koreascience.or.kr/journal/view.jsp?kj=E1JBB0&py=2010&vnc=v6n1&sp=91.

Bishop, M. and Engle, S. 2006. The Software Assurance CBK and University Curricula. :14–21.

Breu, R., Burger, K., Hafner, M. and Popp, G. 2004. Towards a Systematic Development of Secure Systems 2 Basic Concepts of an Object Oriented Software Process.

Freeman, E., Robson, E., Bates, B. and Sierra, K. 2004. *Head first design patterns*.

Hight, S.D. 2005. The importance of a security , education , training and awareness program ( November 2005 ). 27601(November):1–5.

Holmström, A. 2011. Performance and Usability Improvements for Massive Data Grids using Silverlight. . Available http://umu.diva-portal.org/smash/record.jsf?pid=diva2:414906 (accessed 11 May 2013).

Howard, M. and LeBlanc, D. 2009. *Writing secure code*.

International Standards Organization 1989. 7498-2. Information processing systems—Open Systems Interconnection—Basic Reference Model. Part 2: Security Architecture. *ISO Geneva, Switzerland*. Available http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Information+processing+systems+--+Open+Systems+Interconnection+--+Basic+Reference+Model+--+Part+2:+Security+Architecture#0 (accessed 30 April 2013).

International Telecommunication Union 1991. X. 800 Security Architecture for Open Systems Interconnection for CCITT applications. *ITU-T (CCITT) Recommendation*. Available http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:X.800+:+Security+architecture+for+Open+Systems+Interconnection+for+CCITT+applications#0 (accessed 30 April 2013).

Jiang, S., Vaidya, N. and Zhao, W. 2003. Energy consumption of traffic padding schemes in wireless ad hoc networks. *Real-time system security*. Available http://dl.acm.org/citation.cfm?id=903870 (accessed 1 May 2013).

Jones, R. and Rastogi, A. 2004. Secure coding: building security into the software development life cycle. *Information Systems Security*. Available http://www.tandfonline.com/doi/abs/10.1201/1086/44797.13.5.20041101/84907.5 (accessed 4 May 2013).

Jürjens, J. 2002. UMLsec: Extending UML for secure systems development. *≪UML≫ 2002—The Unified Modeling Language*. Available http://link.springer.com/chapter/10.1007/3-540-45800-X_32 (accessed 26 April 2013).

Khan, R.A. and Mustafa, K. 2008. Secured Requirement Specification Framework (S RSF). *American Journal of Applied Sciences* 5(12):1622–1629. Available http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Secured+Requirement+Specification+Framework+(+S+RSF+)#0 (accessed 14 May 2013).

Martin, B., Brown, M., Paller, A., Kirby, D. and Christey, S. 2011. 2011 CWE / SANS Top 25 Most Dangerous Software Errors.

Miller, D. 2003. .NET from the Hacker's Perspective. In: *Black Hat Windows Security 2003 Briefings & Training*. Seattle, WA.

Miller, R.L. 2003. *The osi model: an overview*. SANS Institute.

Nakahara, S. 2000. Electronic Notary System and its Certification Mechanism. *ECIS 2000 Proceedings*. Available http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.104.5051&rep=rep1&type=pdf (accessed 1 May 2013).

National Institute Of Standards and Technology 2011. *Nist special publication 800-53 information security*. 3rd ed. Paramount, CA.: CreateSpace.

Paul, M. 2011. Official (ISC) 2 Guide to the CSSLP. *Software Community (ISC)$^2$ Whitepapers*. Available http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Official(ISC)2+Guide+to+the+CSSLP#0 (accessed 5 May 2013).

Polydys, M.L., Ryan, D.J. and Ryan, J.J.C.H. 2006. Best Software Assurance Practices in Acquisition of Trusted Systems.

Smith, J. 2009. WPF apps with the model-view-ViewModel design pattern. *MSDN magazine*. Available http://www.techguyonline.com/wp-content/uploads/2010/08/wpf_mvvm.pdf (accessed 11 May 2013).

Stoneburner, G., Hayden, C. and Feringa, A. 2001. Engineering Principles for Information Technology Security (A Baseline for Achieving Security), Revision A. . Available http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA393550 (accessed 30 April 2013).

Swanson, M. 1996. Generally Accepted Principles and Practices for Securing Information Technology Systems. (September).

Yi, S., Naldurg, P. and Kravets, R. 2002. A security-aware routing protocol for wireless ad hoc networks. *Urbana*. Available http://www.cis.temple.edu/~wu/teaching/Spring 2013/secure6.pdf (accessed 1 May 2013).

Zhang, C. and Budgen, D. 2012. A survey of experienced user perceptions about software design patterns. *Information and Software Technology*. Available http://linkinghub.elsevier.com/retrieve/pii/S0950584912002297 (accessed 5 March 2013).