# Idea-caution before exploitation: the use of cybersecurity domain knowledge to educate software engineers against software vulnerabilities

**Authors:**

Tayyaba Nafees

Natalie Coull

Ian Ferguson

Adam Sampson

Abertay University, Dundee DD1 1HG

# Idea-Caution Before Exploitation: The Use of Cybersecurity Domain Knowledge to Educate Software Engineers Against Software Vulnerabilities

Tayyaba Nafees, Natalie Coull, Ian Ferguson and Adam Sampson

University of Abertay Dundee, School of Arts, Media and Computer Games. Dundee DD1 1HG `1405357,N.Coull,ian.ferguson,A.Sampson@abertay.ac.uk`
`http://www.abertay.ac.uk`

**Abstract**—The transfer of cybersecurity domain knowledge from security experts ('Ethical Hackers') to software engineers is discussed in terms of desirability and feasibility. Possible mechanisms for the transfer are critically examined. Software engineering methodologies do not make use of security domain knowledge in its form of vulnerability databases (e.g. CWE, CVE, Exploit DB), which are therefore not appropriate for this purpose. An approach based upon the improved use of pattern languages that encompasses security domain knowledge is proposed.

**Keywords:** Software development lifecycle (SDLC), Security pattern (SP), Software Fault pattern (SFP), Attack pattern (AP), Vulnerability database (VDB)

## 1 Introduction

Programmers make mistakes. There are '15-50 errors per 1000 lines of delivered code' (RW.ERROR - Unable to find reference:87). Much research effort has concentrated on addressing this problem (RW.ERROR - Unable to find reference:96). Of particular concern are those software flaws that lead to security vulnerabilities. The deliberate misuse of such a vulnerability is termed an exploitation, resulting in information leaks, and reduce the value or usefulness of the system (RW.ERROR - Unable to find reference:90). Generally, software developers do not understand the security as their focus is on delivering features, rather than on ensuring the software security, so it is often considered as something to be added to a system as a bolt-on component into later stages of development. However, the cost of fixing bugs post software release is estimated to be 30 times pre-release cost (RW.ERROR - Unable to find reference:30). Testing has poor relation with security. It is unusual for the software developer to use testing approaches for finding vulnerabilities; this issue has not received the research attention it requires (RW.ERROR - Unable to find reference:170). One implication of this is that security concerns should be embedded into the software development lifecycle (including the early phases) (RW.ERROR - Unable to find reference:168).

90% of security incidents result from exploitation of flaws in software (RW.ERROR - Unable to find reference:240). In reality, however, software developers struggle against recurring and consistent software flaws (i.e. buffer overflows and integer overflows), which are exploited daily by malicious hackers. Nonetheless, a large body of knowledge about software vulnerabilities exists within the cybersecurity community, in particular amongst penetration testers and ethical hackers. The term 'Ethical Hacker' (EH) will be used as a shorthand to denote this community. Currently ethical hackers put much effort into classifying discovered vulnerabilities and developing taxonomies of these vulnerabilities. Such vulnerabilities are then catalogued in publicly available vulnerability databases (VDBs) (RW.ERROR - Unable to find reference:209). Software developers have worked to embed security within the software development lifecycle (SDLC) (RW.ERROR - Unable to find reference:249) in order to fix the deployment errors. The mechanism of knowledge transfers between the work on vulnerability databases (VDBs), developers' perceptions of security issues and the security development lifecycle (SDLC) is complex, which creates a distinct communication gap between ethical hackers and software engineers (RW.ERROR - Unable to find reference:197). Interception of (knowledge) communication directs software developers to repeat persistent prevalent vulnerabilities and gives rise to software flaws exploitation. Various attempts to capture and formalize the transferring knowledge in a manner appropriate to software engineers have been made, including Misuse Patterns (RW.ERROR - Unable to find reference:81), Software Fault Patterns (SFP) (RW.ERROR - Unable to find reference:69), and Security Patterns (SP) (RW.ERROR - Unable to find reference:227). The need for a better understanding of this mechanism and our proposed solution is the subject of remainder of this paper, which is structured as follows: Section 2 examines previous work in this area and leads the following hypotheses:

**Table 1.** –Proposed Hypotheses

| | |
|---|---|
| **H-1** | Software developers lack the conscious understanding to identify recurring software flaws during software development process due to stagnated and possibly degrading vulnerabilities' knowledge transfer. |
| **H-2** | Patterns (anti-patterns, security patterns and attack patterns) are an appropriate means of communicating knowledge of vulnerabilities from ethical hackers to software engineers. However, existing applications of these pattern languages fail to do so. |

In section 3, shortcomings of previous attempts are analyzed and in section 4, proposals for a pattern-based approach (Vulnerability Anti-Pattern) to the problems are presented.

## 2 Background and Related Work

### 2.1 Building Security by Software Engineers

Other researchers had attempted addressing software developers' security concerns as part of the software development process. For example, earlier attempts have been conducted based upon improving libraries, implementation languages, and language processors (RW.ERROR - Unable to find reference:116, RW.ERROR - Unable to find reference:19). Approaches based on static and dynamic code analysis have been proposed by providing different guidelines, such as SDL banned functions (RW.ERROR - Unable to find reference:188). Software engineers have attempted early exclusion of the vulnerabilities by considering security issues at all phases of the SDLC. Examples of these approaches are considered in Table 2: (RW.ERROR - Unable to find reference:218, RW.ERROR - Unable to find reference:227)

**Table 2.**Approachs To Embed Security In Software Development Processes

| **Name** | **Description** |
|---|---|
| **Security Development Lifecycle (SDL)** | SDL is proposed to reduce software maintenance costs and increase reliability of software with regards to software security related bugs. Cybersecurity standards, such as ISO 27001 are incorporated into the SDL to ensure that any software produced with this process complies with industry recognized standards. However, compliance with standards does not necessarily lead to all vulnerabilities being eliminated from software. The lacking of this model is discussed in section 4.2. |
| **OWASP CLASP** | OWASP Comprehensive, Lightweight Application Security Process includes a set of guidelines for web security requirements, cheat sheets, a development guide, a code review and a testing guide, tools and information about top web security vulnerabilities. This is explored further in section 4.2. |
| **Security Patterns (SP)** | It defines as a solution to stop or mitigate a set of specified threats through certain security mechanisms, and designing to assist software developers who are not security experts with embedding security in their systems. It can also be a useful tool for teaching security concepts (RW.ERROR - Unable to find reference:174). This is explored further in (4.2). However, they are not based directly on the vulnerability knowledge stored in VDBs, which is necessary for achieving currency and a timely response to new threats (RW.ERROR - Unable to find reference:107). |

## 2.2 Attempts by Ethical Hacker to Catalogue and Use Patterns to Communicate Vulnerabilities

The National Vulnerability Database (NVD) comprises CWE, CVE and CAPEC, which are the three most comprehensive vulnerability databases (VDBs). They are open-source and maintained by MITRE (RW.ERROR - Unable to find reference:22) as shown in Table 3. **Table 3**.Attempts to Catalogue vulnerabilities

| **Name** | **Description** |
|---|---|
| **CWE** | The Common Weakness Enumeration database (CWE) catalogues weaknesses that can occur in software. These weaknesses are described as software bugs that can lead to vulnerabilities. |
| **CVE** | The Common Vulnerabilities Enumeration database (CVE) catalogues specific examples of publicly known vulnerabilities that exist in software. |
| **CAPEC** | The Common Attack Pattern Enumeration and Classification database (CAPEC) provides formal attack patterns, while considering the CVE examples and CWE information. |

In addition to the above VDBs, security experts have also endeavored to embed their knowledge of vulnerabilities in the form of patterns (as shown in Table-4) such as SFP, AP and Misuse pattern. This will be explored further in section 4.4.

**Table 4**. Attempts to use patterns to communicate vulnerabilities

| **Name** | **Description** |
|---|---|
| **Software** | SFP is aligned with the CWE database, whose contains a formal specification of weaknesses |

| | |
|---|---|
| **Fault Patterns (SFP)** | (vulnerabilities) and will be explored further in Section 4.4. However, a lack of detailed information about the structure and format of SFP presents a considerable obstacle for software developers. |
| **Attack Patterns (AP)** | AP is derived from CAPEC database, which describes a procedure of a particular vulnerability attack format. However, it is not intended as a source of design patterns (like standard software pattern) Generally, the complicated structure and understanding difficulty restrain developers in their usage. There is not much research done on usage of attack pattern by software developers due to their inherent complexity. |
| **Misuse Patterns** | It describes the malicious hacker generic prospect while considering sub-dimensions, which classifying into set of attack actions and enumerating with possible security patterns as a countermeasure (RW.ERROR - Unable to find reference:80). Although, the misuse pattern groundwork clearly evidences the no usage of cybersecurity knowledge sources (i.e. VDBs) in defining attack action. Thus far, misuse patterns have certain construction deficiencies and lack considerable usage for developers. |

# 3 Analysis

## 3.1 Potential Causes of Poor Knowledge Sharing

The lack of a shared understanding between the Software Engineering and Ethical Hacking communities is well documented (RW.ERROR - Unable to find reference:153, RW.ERROR - Unable to find reference:181). Although there are exceptions, security testing typically takes place as an activity during the SDLC. Ethical Hackers communicate with and report to system administrators and IT managers. Although, there is some crossover between the ground knowledge and skill-set of a software engineer and an ethical hacker, they own some very distinct technical domains, with different educational paths, different technical languages and different professional bodies. Generally, a malicious hacker does not work under the same constraints of project schedules and deadlines as a software engineer does. If they wish to spend six months examining in minute detail of the state of stack under a particular attack condition, they will not have employers pressurizing employees, to deliver. Thus, they have the advantage of time. This coupled with the extensive knowledge sharing that takes place amongst the hacking community (RW.ERROR - Unable to find reference:180) means that a hacker may be more familiar with the weaknesses in a particular piece of software than those who created it.

## 3.2 Software Engineering Problems

The approaches from section 3.1 are attempts by the software engineering community to enable the integration of security concerns into the process of developing software. The approaches, such as SDL, OWASP CLASP and SP, focus only on fulfilling security guidelines and standards rather than raising awareness of vulnerabilities. SDL does not embed any knowledge from cybersecurity experts and are challenging for those software developers who have limited awareness and understanding of the security vulnerabilities in order to apply the security guidelines effectively. The organizational emphasis of SDL may also be of limited applicability in the informal world of cross platform application deployment. OWASP CLASP implementation is limited to web-based systems. Furthermore, the value of SPs in order to provide usable and understandable documentation for developers is questionable due to their complexity (RW.ERROR - Unable to find reference:285), and they are generally not adopted by developers due to their poorly described implementation (RW.ERROR - Unable to find reference:41). This can be attributed to the lack of an accepted standard catalogue and a lack of methodological support.

## 3.3 Cyber Security Problems

The various databases described are maintained by cybersecurity professionals to keep track of known vulnerabilities in the different versions of released software. It is clear that the intended audience for these databases is not software engineers involved in developing software but rather systems administrators looking to secure their existing systems. It might be possible that the information contained therein is simply not generalized enough to be directly relevant for software developers to use in the development process. Some of the difficulties that software developers face are enumerated in Table-5: **Table 5.** VDBs issues

| | |
|---|---|
| **No standardization** | No standard taxonomy/classification scheme for existing VDBs, thus each of them use their own approach, none of which were explicitly designed to use during SDLC. As such, these VDBs can typically appear complex and ambiguous to the software developer (RW.ERROR - Unable to find reference:142, RW.ERROR - Unable to find reference:150, RW.ERROR - Unable to find reference:134). |
| **Limited knowledge** | Closed source VDBs, such as the Carnegie-Mellon US Cert database and Secunia, are of necessity limited in the information that they can show concerning code-level errors. |
| **Complexed knowledge** | It is clearly shown by many research studies, which have compared vulnerability information across the multiple VDBs that these repositories are deficient in providing interoperability, knowledge consistency and are not following standard classification schemes (RW.ERROR - Unable to find reference:207, RW.ERROR - Unable to find reference:208). |

## 3.4 Addressing Shortcomings of Previous Pattern-Based Attempts

Section 3.3 discussed previous attempts to use patterns/pattern languages in the cybersecurity context. These attempts highlighted the following shortcomings: a distinct communication gap between software developers and ethical hackers; software developers lack conscious understanding about prevalent vulnerabilities because of unusable and complicated knowledge sources, SDLC does not adequately address software security practices, and finally there are limited efforts from both the cybersecurity and software engineering communities to work together to address software vulnerabilities. It is clear that the use of patterns can only succeed in the context of an appropriate software development process, which must include knowledge from the VDBs. The author's future work will examine the way in which patterns can be used to capture VBDs knowledge in a usable format, the need to provide understandable vulnerabilities' awareness to developers is emphasized by Fahl et.al and Acar et.al work (RW.ERROR - Unable to find reference:286, RW.ERROR - Unable to find reference:287) . The desirability of a methodology and tool is also support by McGraw (RW.ERROR - Unable to find reference:150) and Borstad (RW.ERROR - Unable to find reference:196).

## 4 Practical Proposition: Our Solution

To address these issues, our research has led to the creation of a set of ‘Vulnerability Anti-Patterns’, based on the OWASP Top 10 Vulnerabilities. Our anti-patterns have been constructed following two main stages: knowledge extraction (1-knowledge pulling process sourced from VDBs and security patterns) and knowledge provision (2-knowledge pushing process to educate developers through anti-patterns).

### 4.1 The Knowledge Extraction (1-Knowledge Pulling Process)

The knowledge pulling process sourced by cybersecurity community such as VDBs (CWE, CVE), security patterns and attack pattern databases (CAPEC), and collected essential information of the vulnerability. For example, general information, root-causes and attack procedures. This is the first step towards addressing the communication gap. The knowledge pulling process comprises two sub-parts: 1) Creating a taxonomy of vulnerabilities. The taxonomy includes vulnerability info, vulnerability footprints or characteristics and mitigation categories; 2) generating a decision tree which describes the vital VDBs information, and shows safeguard and injury paths that map security incidents with their low-level and high-level root-causes of vulnerabilities in respective phase of software development life cycle (SDLC).

### 4.2 Knowledge Provision (2-Knowledge Pushing Process)

Extracted knowledge passed to the knowledge pushing process, which captures previous process formalized information in the form of patterns, known as Vulnerability Anti-Patterns that is most appropriate mechanism to communicate knowledge of vulnerabilities to software developers.

#### 4.2.1 The Notion of Vulnerability Anti-Pattern (VAP)

A recurring error or vulnerability initiates an anti-pattern, which can occur due to any poor software design or implementation errors. Same in the case of vulnerabilities, which are, commonly reoccurring flaws, so why does not capture and address the fundamental problems of cybersecurity through anti-patterns. A VAP describes a problem, i.e. poor practice that negatively causes a security flaw, and a solution, i.e. a set of refactoring actions that can be carried out to mitigate or stop flaws. In contrast to SP, which are only designed to perceive a threat, not to repair a vulnerability, and VDBs that appear complicated for developers’ understanding and are generally not considered as a part of developers’ security practices. It has been argued (RW.ERROR - Unable to find reference:172) that the prevalent software errors occurred because of established software practices that actually have negative impact during SDLC. Such poor practices generally cause prevalent vulnerabilities. It can thus be suggested that these poor practices need to be identified and refactored so safe solutions can be generated (RW.ERROR - Unable to find reference:9). The use of anti-patterns for finding and understanding vulnerabilities is understudied, particularly for software developers. VAP can describe poor practices or solutions, which aid in reasoning about and communicating unsuccessful design intent, and introduce refactored solutions, which suggests safe alternative procedures. The advantage of adopting VAP during software development process is that it bridges the knowledge gap between software developers and security experts about commonly occurring software flaws. This finding has important implications for developing security training methods. Therefore, an anti-pattern is suggested for the vulnerability that includes necessary vulnerability information in a well-defined and usable format for those inexperienced and naive developers who do not understand security and can be an effective way of communicating vulnerable poor practices, so developers can learn valuable lesson from other fellows' successes and failures. Without this wisdom, anti-patterns of prevalent vulnerabilities will continue to persist.

**Vulnerability Anti-Pattern: A Proposed Solution.**

Authors propose a new refactored solution called 'Vulnerability Anti-Patterns' that are intended to provide the developers' security necessary awareness. Since the vulnerability anti-patterns' core objective is to highlight the entire software exploitation potential, each pattern has been written to describe the following: general practices of the anti-pattern (i.e. how it could be misused), examples such as CVE (real-world exploitation) and sample vulnerable code, and finally the footstep of risk patterns within SDLC, the refactored solution and related solutions in the form of security patterns. Ultimately, the anti-patterns should enable the developers to realize the root-cause of the vulnerability. In regards to the proposed solutions (or countermeasures to the vulnerabilities), we anticipate that the anti-patterns will encourage the developers to retain a deep understanding and conscious alertness of vulnerabilities in their future development practices. Our template for an anti-pattern is presented below. We have utilized this template and produced complete anti-patterns for 10 vulnerabilities to date. In addition to the complete pattern data outlined below, we have also produced an abridged version of each pattern, which describes, using languages from various different programming languages how each vulnerability can be exploited. **Vulnerability Anti- Pattern Template. Table 6.**

| Pattern Main-Division | Pattern Sub-Division |
|---|---|
| **1. Vulnerability Anti-Pattern General info** | **1.1. Anti-Pattern Name:** |
| | **1.2. Also Known as:** |
| | **1.3. Most Frequent Scale in SDLC: Requirement Specification, Design, Implementation/Coding-Phase** |
| | **1.4. Problem Description:** |
| | **1.5. CWE Mapping: CWE-ID, General Name** |
| | **1.6. Related CWEs:** |
| | **1.7. CVE Example:** |
| **2. Anti-Pattern (Problematic Solution)** | **2.1. Refactored Solution Name:** |
| | **2.2. Refactored Solution Type: Software Pattern, Technology Pattern, Process Pattern, Role Pattern** |
| | **2.3. Root Causes (Context): Unbalanced Forces related to meeting requirements, controlling technology changes, controlling use and implementation of people.** |
| | **2.4. Risk patterns and Consequences:** |
| | **2.5. Typical Causes** |
| **3. Problem Fingerprints** | **3.1. Software Fault Pattern (SFP)** |
| **4. Known Exploitation** | **4.1. Attack Pattern (Attack patterns-CAPEC)** |

| | |
|---|---|
| **5. Mitigation (Refactors the problem)** | **5.1. Refactored Solutions:**<br>**5.1.1. Solution Steps SDLC, Description**<br>**5.2. Examples: (Real-world Patch Example)** |
| | **5.3. Pen Testing Techniques** |
| | **5.4. Related Solutions(SP):**<br>**5.4.1. General Solution (All in one solution)** |
| | **5.4.2. Language Solution** |

### 4.3 Evaluation

To evaluate the effectiveness of ‘Vulnerability Anti-Patterns, we are in the process of conducting a series of experiments with software developers from two international organizations and computing students from our own university. **Stage-1:** Pre-assessment evaluation to measure participants’ actual awareness about poor development practices. **Stage-2:** Participants are trained while using informal versions of the vulnerability anti-patterns. **Stage-3:** Post assessment evaluation to how much participants able to improve their understanding of vulnerabilities accompanied by the formal version of vulnerability anti-patterns. **Stage-4:** Comparative analysis performed between trained and untrained developers to measure the progression in developers’ abilities to identify and understand the most commonly persistent software flaws regarding the efficiency of vulnerability anti-patterns.

## 5 Conclusion

Secure software development is one of the most challenging areas of cybersecurity. Although, the cybersecurity industry is mature and generates a wealth of resources on discovered software vulnerabilities in the form of VDBs, software developers are continuing to produce recurring and persistent software flaws at an alarming rate. The software engineering community has also worked to embed security into the SDLC; however, these independent efforts fail to provide effective solutions against prevalent vulnerabilities. Hackers on a daily basis exploit a large number of fatal development errors. Software developers are largely unaware of the design and implementation-level security flaws (poor development practices) which generally turn into fatal security weaknesses (vulnerabilities). There exists a big communication gap between the software developers and security experts, which does not help them to solve this problem. The research proposes a methodology to use a pattern for transferring a necessary vulnerabilities knowledge to software developers through ‘Vulnerability Anti-Pattern’, and considers the use of patterns to communicate knowledge of software vulnerabilities in usable format with the best means of avoiding their creation. It bridges the communication gap between them with assistance of classified cybersecurity knowledge sources such as VDBs, which ultimately share essential information about common errors and help to identify software developers’ secure ideas to build secure software. We propose that one solution to this problem lies in the use of patterns languages (with appropriate methodological, tool and training support) to better capture and communicate the information currently held in VDBs to create a ‘Safe Development Environment’. Therefore, knowledge of vulnerabilities can bridge the communication gap between cybersecurity and software engineering communities. It is toward this goal that our future work, based upon this initial study will be directed.

## References Reference list