Vulnerability Anti-Patterns: Essential Education for Software Developers

Tayyaba Nafees, Natalie Coull, Ian Ferguson, Adam Sampson | Abertay University, Dundee

Abstract: As there exists a big communication gap between software engineering and cybersecurity communities, this prevents them to share necessary security knowledge. Anecdotally, the cyber security community sources (i.e. vulnerability Databases (VDBs), Common Weakness Enumeration (CWE) are the best means of acquiring secure software development information, however their captured knowledge form is complicated for developers' understanding. The transfer of cybersecurity domain knowledge from security experts ('Pen-Testers') to software engineers is discussed in terms of desirability and feasibility. A methodology is proposed based upon the improved use of pattern languages, which sourced cybersecurity community knowledge to educate software developers against prevalent vulnerabilities.

Introduction:

The Internet Security Threat Report by Symantec Corporation [1] states 6.4 billion internet-connected devices are mutilated by thousands of estimated cyber-attacks globally every second. For example JPMorgan Chase data breach [2], Sony [3] and The Home Depot data breach [4] . Questions arising from this are:

- Why do malicious hackers know more about our systems than developers and security experts?
- What are the knowledge gaps or disconnects between developers and security experts that allow malicious hackers to succeed?

90% of security incidents result from exploitation of flaws in software systems [5]. The unfortunate reality is that software developers struggle against these recurring and consistent software flaws (commonly known as vulnerabilities) such as buffer overflows and integer overflows, which are exploited by hackers on a daily basis. Nonetheless, there exists a wealth of cybersecurity domain knowledge about software vulnerabilities and errors held in vulnerability databases (VDBs), such as National Vulnerability Database (NVD)[6], CWE [7], Common Vulnerabilities and Exposures (CVE) [8] and Common Attack Pattern Enumeration and Classification (CAPEC) [9]. Despite the software engineering community's purposive endeavours and the cybersecurity community's best efforts, the number of dangerous software exploitations are increasing at alarming rate [10, 11]. It is thus clear that these resources are not being utilised effectively in providing necessary knowledge to the software developer due to the following reasons: information overload, lack of techniques to systematically annotate flaws' rectification and insufficient analysis of the data relating to these vulnerabilities [12-14]. The use of anti-patterns for finding and understanding vulnerabilities is understudied, particularly for software developers.

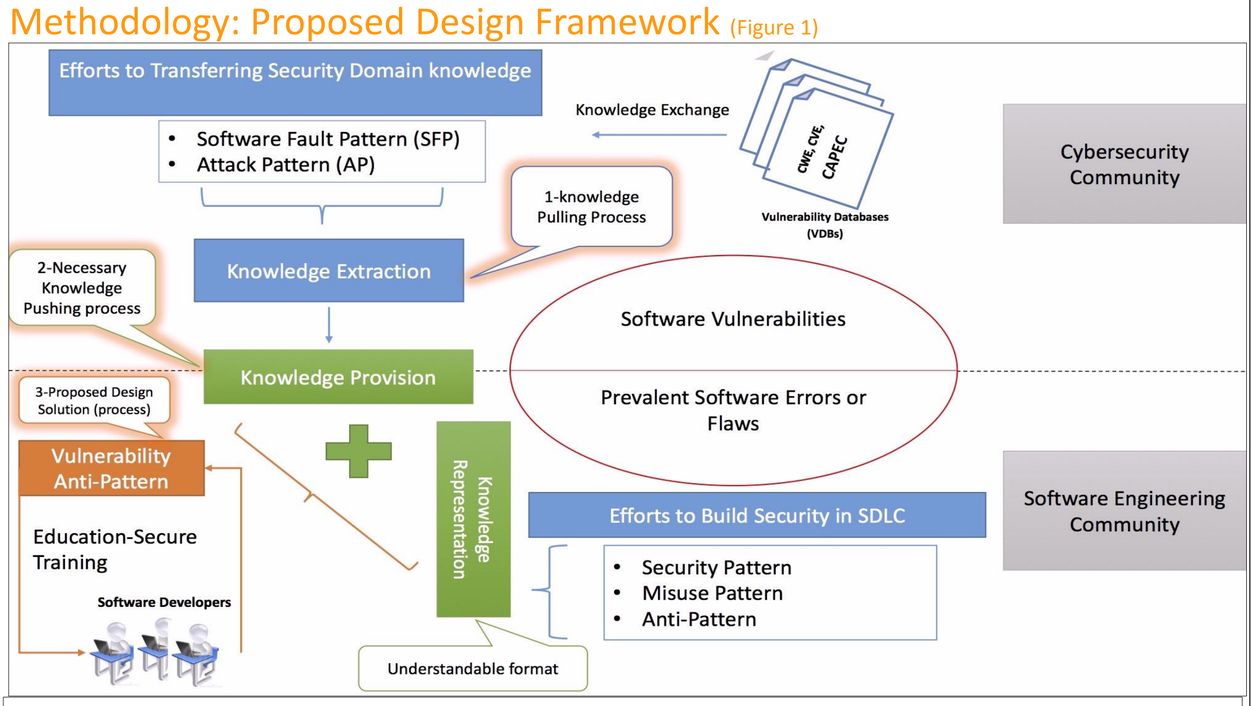
Research Question

Do software developers have conscious knowledge (essential understanding) of prevalent vulnerabilities and do they know how to mitigate them during the software development life cycle (SDLC)?

Hypotheses

- Software developers lack the conscious understanding to identify recurring software flaws during development process due to stagnated and possibly degrading vulnerabilities' knowledge transfer.
- The cyber security community sources (i.e. vulnerability databases, attack patterns, software fault patterns) are the best means of acquiring secure software development information, however their captured knowledge form is inadequate and incomprehensible for developers' understanding.
- Vulnerability Anti-Patterns (VAP) have been designed to bridge the awareness gap by providing considerate vulnerabilities' knowledge, which included the scrutinized knowledge from the cybersecurity community sources to educate software developers against prevalent vulnerabilities.

Methodology: Proposed Design Framework (Figure 1)



To address these issues, the authors propose a methodology (Figure 1), which 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 in three main stages: knowledge extraction, knowledge provision and education.

1. Knowledge Pulling Process: Knowledge Extraction

- Necessary knowledge pulling from vulnerability databases (VDBs) and security patterns (SP); this process involves the act of knowledge exploration and examination as shown in Table 1.

2. Knowledge Pushing Process: Knowledge Provision

- This process makes use of anti-patterns to capture and integrate the extracted information, which is concluded from knowledge pulling process, so software developers have access to the distilled wisdom of experts in dealing with recurring security problems.

3. Education Process: Vulnerability Anti-Pattern

- To Educate software developers through proposed design solutions (called Anti-Patterns) is critically important; its evaluation process is carefully designed to measure effectiveness and usability in regard to developers' abilities to identify and understand the root-causes of vulnerabilities. Table 1: Knowledge Extraction

Vulnerability Level	Software Fault Pattern	Vulnerability Databases (VDB)	Analysis Process of Security Patterns
Requirement Level Vulnerabilities	Injury, Safe guard	CWE	Security Objectives
Design Level Vulnerabilities	Injury, Safe guard	CWE,CVE	Design Patterns/Anti-Pattern
Code/Implementation Level Vulnerabilities	Injury, Safe guard	CAPEC	Security Coding/Attack Pattern

1. Knowledge Pulling Process: The Knowledge Extraction

- **Taxonomy of Vulnerabilities:** Generating a taxonomy of information in a well-defined format that is sourced from CWE, CVE, CAPEC and security patterns, consisting of the following information: vulnerability info, vulnerability footprints or characteristics and mitigation. The vulnerability taxonomy is demonstrated with an example of CWE-89 [7,8,9] in Table 2.
- **Decision Tree:** A sub-process of the knowledge extraction to map the association among security incidents with their low-level and high-level root causes in the software development life cycle (SDLC) phases as shown in Figure 2. Green paths are safeguards that direct the developer to avoid software errors; Following Red (injury) paths can lead to the creation of a vulnerability.

2. Knowledge Pushing Process: The Knowledge Provision

- **The Analogy between Anti-Pattern and Vulnerabilities:** For a software engineer, Design pattern generally considers a good practice; anti-pattern is a poor practice. However, sometimes a good turn into a bad, similarly, design patterns can turn into anti-patterns. The result is that, a pattern that may be commonly used and generally considered as a good practice, but now it is ineffective and counterproductive in practice. Thus we developed a novel approach called a "Vulnerability Anti-Pattern", which encapsulates root causes of vulnerabilities/recurring software flaws. It accentuates the realisation that vulnerability anti-pattern can address vulnerabilities' exploitation and their related threats more effectively than local fixes.

3. Education Process: Vulnerability Anti-Pattern

- **The Notion of Vulnerability Anti-Pattern:** A refactored solution, intended to stimulate the developers' awareness, so they will apprehend the extracted security knowledge. To achieve this, the vulnerability anti-pattern, whose core objectives highlights the entire software exploitation perspective, resides in five main divisions as described in Table 3.

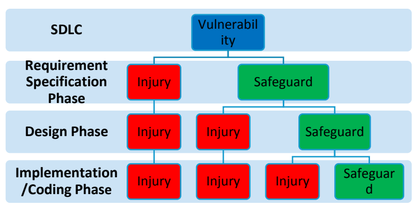
Table 2: CWE-89

Vulnerability info			Vulnerability fingerprints or Characteristics			Mitigation		
CWE-ID	CVE	Generally known as	Context	Lifecycle	SFP	STRIDE	SP	AP
CWE-89	CVE-2016-1393CVE-2015-0161CVE-2008-5817	SQL Injection	Developing a system with database-driven web sites and save user inputs in a database.	Design Phase	CWE-990: SFP Secondary Cluster: Tainted Input to Command	Spoofing	Intercepting Validator	CAPEC-7 CAPEC-66 CAPEC-108 CAPEC-109 CAPEC-110

Table 3: Template of VAP

Pattern Main-Division	Pattern Sub-Division	
1.	**Vulnerability Anti-Pattern General info**	1) Anti-Pattern Name: 2) Also Known as: 3) Most Frequent Scale in SDLC: Requirement Specification, Design, Implementation/Coding-Phase 4) Problem Description: 5) CWE Mapping: CWE-ID, General Name 6) Related CWEs: 7) CVE Example:
2.	**Anti-Pattern (Poor Practices)**	1) Refactored Solution Name: 2) Refactored Solution Type: Software Pattern, Technology Pattern, Process Pattern, Role Pattern 3) Root Causes(Context): Unbalanced Forces related to meeting requirements, controlling technology changes, controlling implementation of people. 4) Risk patterns and Consequences: 5) Typical Causes
3.	**Problem Fingerprints**	1) Software Fault Pattern (SFP)
4.	**Known Exploitation**	1) Attack Pattern (Attack patterns-CAPEC)
5.	**Mitigation (Refactors the problem)**	1) Refactored Solutions: Solution Steps during SDLC; Description 2) Examples: (Real-world Patch Example) 3) Pen Testing Techniques 4) Related Solutions(SP): General Solution (All in one solution), Language Solution

Figure 2: Decision Tree



Results: Evaluation

To evaluate our method, we recruited participants (software developers) from graduate level students from **the School of Arts, Media and Computer Games (AMG) at University of Abertay Dundee** and performed the pilot experiment study with them. The results from the pilot study are promising. However, evaluation in the real world with industry software developers is in-progress.

Evaluation Phase-1: Interview and Observation

By doing informal interviews with a number of software developers and Pen-Testers from industry, the following conclusions have been drawn.

- Q1-Do software developers and penetration-testers share evaluation log of errors/vulnerabilities information, static analysis tools (SAC) results and commonly occurring errors/vulnerabilities statistics with each other?
- Answer: 4 out of 5 said No.
- Q1-Do software developers and penetration-testers have communal understating in describing the vulnerability (do you have a common understanding language i.e. design flaws or vulnerable code)?
- Answer: 3 out of 5 said they do not have any common understanding

Evaluation Phases-2&3: Experiment Phase & Education Phase

After considering 'CWE/SANS Top 25 Most Dangerous Software Errors'; the experiment questionnaire was specially designed to elicited developers' awareness; 1) vulnerable code or UML diagram; 2) investigated vulnerability root-cause description; 3) explored related attack patterns and exploitation. Both phases are interlinked and occur in 4 stages.

1.Stage 1: Pre-Assessment Survey Study

Participants completed a questionnaire consisting of open-ended and closed-ended questions. The detail of included vulnerabilities and their related "Vulnerability Anti-Patterns" is described in Table 4

2.Stage 2: Security Training Session (Education Phase)

Participants received security training through "Vulnerability Anti-Patterns" against a set of included vulnerabilities.

3.Stage 3: Post-Assessment Survey Study

Participants completed the questionnaire again; question syntax was changed.

4.Stage 4: Post-Post-Assessment Survey Study (after a one week gap)

Participants were asked to complete stage 4, to measure how much they were able to retain the information from the security training provided.

Preliminary Results

The preliminary results of the experiment in Figures 4 and 5 show that, before security training, participants struggled with vulnerable code and Misuse UML class or sequence diagrams (maximum score was 10/15). However, After security training, participants scored higher, as the stage 2 minimum score was equal to maximum score of stage 1, which clearly indicates they managed to understand the vulnerabilities. From the statistical descriptive analyses of both stages scores (TOTAL_stage1 and TOTAL_stage2) mean values and standard deviation values that participants performed better after receiving the training. Overall, these results indicate that security training method had significantly improved participants' ability to identify and understand the root causes of vulnerabilities during different stages of the development process.

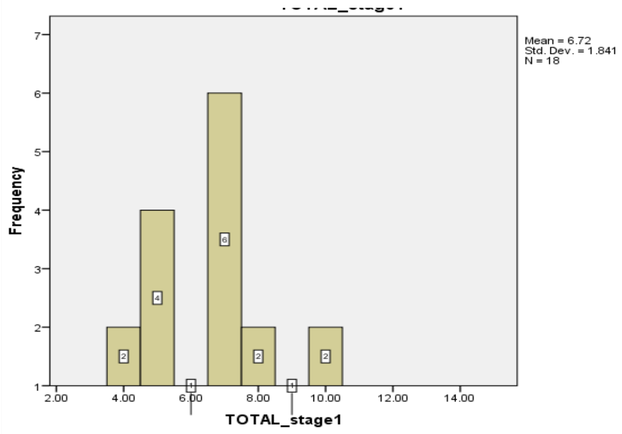


Figure 4: Score Before Training

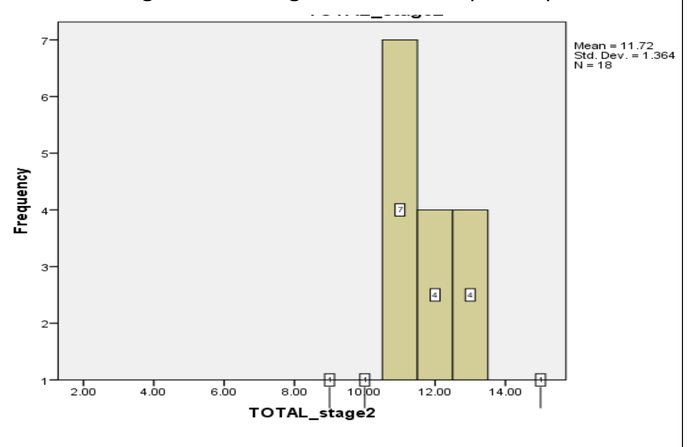


Figure 5: Score After Training

Conclusion:

The construction of secure software is one of the most challenging task for developers because of the growing sophistication of security requirements, and the progress of attack vectors. Clearly, a strong need exists to education education of software developers about those commonly occurring flaws that occurs because of overlooked software's security practices and causes exploitation. The research proposes a methodology to transfer a essential vulnerabilities awareness to software developers through 'Vulnerability Anti-Pattern', and proposes the use of patterns to communicate knowledge of software vulnerabilities with the best means of avoiding their creation. It bridges the communication gap between them (pent-testers and software developers) with assistance of organized 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 (with appropriate methodological, tool and training support).

Table 4: Detail of Included Vulnerabilities in the Experiment Study

#	Vulnerability Anti-Pattern	Computing	Gaming	Example code language	Vulnerability
1	SQL Injection Vulnerability Anti-Patter	√		PHP	SQL Injection
2	Missing Authentication for Critical functions Vulnerability Anti-Pattern	√		PHP	Missing or Broken Authentication
3	Missing Authorization Vulnerability Anti-Pattern	√	√	UML class diagram and sequence diagram	Missing or Broken Authorization
4	Buffer Overflow Vulnerability Anti-Pattern		√	C/C++	Buffer over flow
5	Use of Deprecated Function Vulnerability Anti-Pattern	√		PHP	Dangerous function calls
6	Use of Potentially Dangerous Function Vulnerability Anti-Pattern		√	C/C++	Dangerous function calls
7	Integer Overflow Vulnerability Anti-Pattern	√	√	PHP, C/C++	Integer overflow
8	Incorrect Calculation of Buffer Size Vulnerability Anti-Pattern		√	C/C++	Incorrect Calculation of Buffer Size

References

[1] Symantec Corporation, "2016 Internet Security Threat Report," Symantec Corporation., Tech. Rep. 04/16 21365088, 2016.

[2] J. Silver-Greenberg, M. Goldstein and N. Perlroth, "JPMorgan Chase Hack Affects 76 Million Households," New York Times, vol. 2, 2014.

[3] B. Schneier, "Attributing the Sony Attack," Retrieve From, 2015.

[4] B. Hawkins, "Case Study: The Home Depot Data Breach," 2015, vol. SANS Institute InfoSec Reading Room, 2015. 2015.

[5] DHS, "Cyber Incident Response at DHS," Https://Www.Dhs.Gov/Blog/2016/07/28/Cyber-Incident-Response-Dhs, vol. 2017, 2017.

[6] MITRE Corporation, "Common Weakness Enumeration," 2015.

[7] MITRE Corporation, " Common Vulnerabilities and Exposures (CVE)," Exposures (CVE), 2015.

[8] MITRE Corporation, "Common Attack Pattern Enumeration and classification," 2014.

[9] T. Shiralkar and B. Grove, "Guidelines for Secure Coding," Atsec Information Security Corporation, Technical Report, 2009.

[10] Y. Wu, I. Bojanova and Y. Yesha, "Reconstruction of common weakness enumeration," in SI]: IEEE Software Technology Conference (STC), pp. 28, 2014.

[11] A. Arnold, B.M. Hyla and N.C. Rowe, "Automatically building an information-security vulnerability database," in Information Assurance Workshop, 2006 IEEE, pp. 376-377, 2006.

[12] H. Ghani, J. Luna, A. Khelil, N. Alkadri and N. Suri, "Predictive vulnerability scoring in the context of insufficient information availability," in Risks and Security of Internet and Systems (CRiSIS), 2013 International Conference on, pp. 1-8, 2013.

[13] G. Yun-hua and L. Pei, "Design and Research on Vulnerability Database," in 2010 Third International Conference on Information and Computing, pp. 209-212, 2010.