Towards Effective Security Assurance for Incremental Software Development The Case of Zen Cart Application

Azmat Ali
Fraunhofer SIT
Darmstadt, Germany
azmat.ali@sit.fraunhofer.de

Lotfi Ben Othmane
Fraunhofer SIT
Darmstadt, Germany
lotfi.ben.othmane@sit.fraunhofer.de

***Abstract*—Incremental software development methods, such as Scrum embrace code changes to meet changing customer requirements. However, changing the code of a given software invalidates the security assurance of the software. Thus, each new version of a given software requires a new full security assessment. This paper investigates the impact of incremental development of software on their security assurances using the e-commerce software Zen Cart as a case study. It also describes a prototype we are developing to design security assurance cases and trace the impact of code changes on the security assurance of the given software. A security assurance case shows how a claim, such as "The system is acceptably secure" is supported by objective evidence.**

***Index Terms*—agile, security assurance case, software security.**

I. Introduction

Agile software development methods, such as Scrum are widely used to incrementally and iteratively develop software. The iterative nature of these methods supports frequent code changes in order to meet changing customer requirements. The common approaches for software security assurance (ensuring that the software fulfills its security requirements) focus on ensuring that the security properties are enforced for a given version of software. They have a significant cost associated with their use in time and resources [1], [2].

Security assurance enables developing coherent objective argumentation that supports claiming that a software mitigates its security risks [3]. The evolution of software in terms of frequent code changes in successive iterations could make the implemented security measures and defined security tests ineffective or incomplete for subsequent iterations of the software. Thus, identifying the impacts of code changes on the security of the software is critical to the maintenance of security assurance of the given software[3].

We explore in this paper how code changes impact security assurance in the context of Zen Cart.[1] Zen Cart is a PHP-based (Web-based) e-commerce open source software. First, we analyzed the vulnerabilities of Zen Cart reported in the NVD database.[2]. Then, we interviewed the security lead of Zen Cart to learn the applied techniques for addressing security assurance in the context of incremental software development.

In previous work we proposed the use of security assurance cases as a representation of the security of the software and the maintenance of the cases according to the code changes [4], [3]. A *security assurance case* presents arguments showing how a top-level claim (e.g., "The system is acceptably secure.") is supported by objective evidence [5]. We describe in this paper the Eclipse plug-in that we are developing to design security assurance cases and to trace the impact of code changes on the assurance cases.

The paper is organized as follows. Section II discusses related work. Section III gives an overview of Zen Cart and describes the analysis of Zen Cart vulnerabilities, the threat modeling, and the penetration testing that we performed. Section IV provides the results of the interview we had with the security lead of the Zen Cart. Section V describes the prototype that we are developing for designing security assurance cases and tracing code changes to the artifacts of security cases of the given software. Section VI concludes the paper.

II. Related Work

Change Impact Analysis (CIA) determines the effects of software changes [7]. There has been extensive work on CIA from software behavior perspective [8]. For example, Xiaoxia et al. [9] developed Chianti, a change impact analysis tool for Java. The tool analyzes two versions of an application and decomposes their differences into a set of atomic changes. Then, it identifies the impacts of changes in terms of affected tests; it identifies execution behavior that may have been modified by the changes.

Traceability and dependency analysis have effectively supported software impact analysis for software behavior. However, as software grow in size and complexity, their dependency webs of information extend beyond software engineers' abilities to comprehend. Bohner et al. [10] examined extending software change impact analysis to incorporate interoperability and dependency relationships for addressing distributed ap-

[1] http://www.zen-cart.com/
[2] https://nvd.nist.gov/

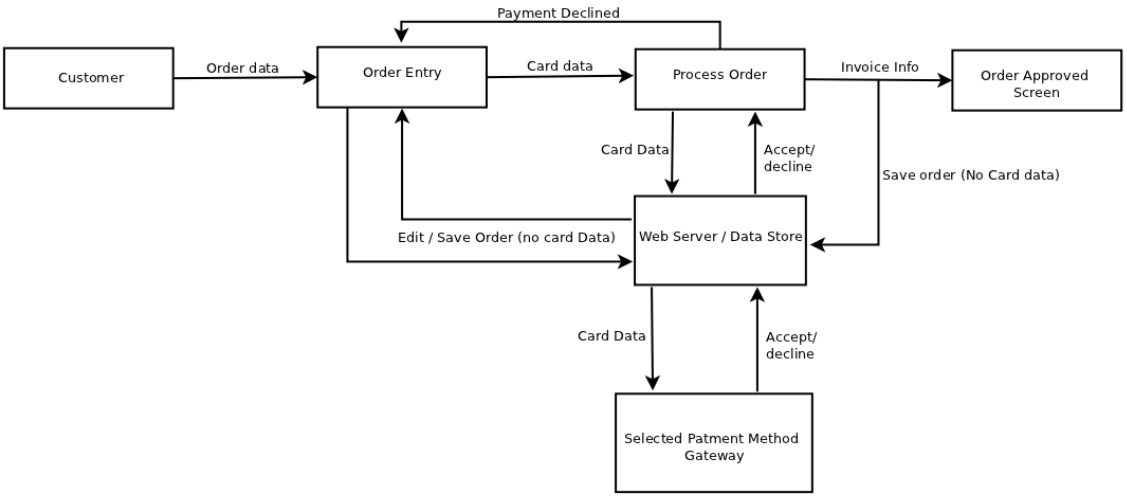

Figure 1. UML Collaboration diagram for Zen Cart [6]

plications and explored 3D visualization techniques for more effective navigation of software changes.

Common approaches for software security assurance focus on ensuring that the security properties are enforced during the initial system development. They have a significant cost associated with their use in time and resources [1]. There has been extensive investigation of the impact of changes on the security of software at design levels, such as the secure software evolution project [11]. Jurjens et al. [1], for example, proposed extending UML models with security annotations, which they named, UMLSec. They investigated the security analysis of UMLSec models by means of a change-specific notation allowing multiple evolution paths and sound algorithms supporting the incremental verification process of evolving models. Also, Raschke et al.[12] proposed an agile security evaluation method for the Common Criteria standard. The method allows the management of changing requirements and provides feedback regarding software security.

Othmane et al. [3] proposed integrating security reassurance into the agile software development processes to ensure the security of the developed software increment in each iteration. They, further demonstrated the use of the technique to iteratively develop security features that fulfill their security requirements [4]. The process helps, for example, to identify customer change requests that conflict with the security requirements of the iteration.

This work extends our earlier work [4], [3] by analyzing the impacts of incremental secure software development for an existing open source software, Zen Cart. It also describe the prototype we are developing to implement the solution we provided in [4], [3].

III. Overview of the Security of Zen Cart

This section describes the analysis of the Zen Cart vulnerabilities that were reported in the National Vulnerability Database (NVD)[3], and the threat modeling that we performed using the available collaboration diagram of Zen Cart.

A. Overview of the application

Figure 1 shows the collaboration diagram of Zen Cart [13]. Zen Cart can be used to setup customized online shopping stores. A customer uses the application to choose among different products, order the selected products, pay the order, and get an invoice. Zen Cart manages the full process, from maintaining the products catalogue to the delivery of the purchased products to the customers. The application validates the payments so the vendor can ship the orders to the customers' addresses.

Zen Cart is a PHP-Based open source. It uses MySQL database for data storage and runs on most of the Web servers that support PHP. A typical deployment requires isolating the Web application and the database (including the database management system) through a firewall. The application must communicate with the payment gateways through HTTPS protocol. The customers also access the application using the HTTPS protocol.

We used PHP Pear module[4] to extract the XML Metadata Interchange (XMI) representation (A standard format for exchanging data between sofwtare development tools.) from the Zen Cart source code. Then we used ArgoUML [14] to extract the class diagram from the the XMI data. We obtained a big set of unrelated classes that we couldn't use further to derive a higher-level diagram such as a component diagram.

B. Analysis of Known Zen Cart vulnerabilities

Code changes, in Zen Cart, are managed through github, a version control system. The first version of Zen Cart was insecure: the team received constantly vulnerability reports from

[3]https://nvd.nist.gov/
[4]https://pear.php.net/

Table I
ZEN CART VULNERABILITIES REPORTED IN NVD.

No	Vulnerability type	Level	Total	Versions	CVSS
1.	SQL Injections	Code	07	1.1.2d to 1.3.8a	5.1 to 7.5
2.	Cross-site scripting	Code	07	1.3.5 to 1.5.1 (ja)	2.6 to 6.8
3.	Data Validation	Design	04	Note 3	5.8
4.	Authentication Issues	Design	03	1.3.7, 1.3.8, 1.3.8a and earlier	6.8 to 8.5
5.	Information Leak/Disclosure	Design	03	1.2.6 and earlier, Note 3	2.6 to 7.5
6.	Path Traversal	Design(2), Code(1)	03	1.3.0.2, 1.3.8 and 1.3.8a	6.8 to 7.5
7.	Input Validation	Design	02	1.3.0.2, 1.3.8 and 1.3.8a	5.0
8.	Code Injection	Design	02	1.3.0.2 and earlier	5.1
9.	Permissions	Design	01	1.1.0 to 1.2.6d	10.0
10.	Cross-site request forgery	Code	01	1.3.9h	5.8
11.	Code	Code	01	1.1.4	7.5
12.	Unknown	Unknown	01	1.2.7 and earlier	10.0

Notes:
1: CVSS stands for Common Vulnerability Scoring System.
2: Ja means Zen Cart Japanese version.
3: Zen Cart versions are not specified by NVD for the specified vulnerabilities.

Table II
THREAT MODELLING RESULTS FOR ZEN CART.

No	Threat Category	Threat	Total
1.	Spoofing	Spoofing of Destination Data Store / Web Server	07
2.	Information disclosure	Weak Access Control for a Resource	03
3.	Denial of Service	Potential Excessive Resource Consumption for Web Service or Data Store / Web Server	04
4.	Tampering	Process Order / Order Entry Process Memory Tampered	03
5.	Privilege escalation	Elevation Using Impersonation	04
6.	Spoofing	Spoofing the Customer External Entity / Order Entry Process	02
7.	Tampering	Potential Lack of Input Validation for Order Entry	01
8.	Repudiation	Potential Data Repudiation by Order Entry	01
9.	Information disclosure	Data Flow Sniffing	01
10.	Denial of Service	Potential Process Crash or Stop for Order Entry	01
11.	Denial of Service	Data Flow Order data Is Potentially Interrupted	01
12.	Privilege escalation	Order Entry May be Subject to Elevation of Privilege Using Remote Code Execution	01
13.	Privilege escalation	Elevation by Changing the Execution Flow in Order Entry	01

security companies. The team fixes reported vulnerabilities and includes the updates in the new versions of the software. The users and developers are informed about released security patches, so publicly known vulnerabilities will not affect them.

Discovered vulnerabilities in released Zen Cart versions are published to the NVD and are given a Common Vulnerabilities and Exposures (CVE) number. We identified 35 vulnerabilities that were published between December 2004 and April 2015. Table I gives a summary of the identified vulnerabilities grouped by vulnerability type. We observe that many of reported vulnerabilities were of type SQL injection (7) or XSS (7) and that about half of the reported vulnerabilities were design flaws–the other half were code-level flows. We understand that even the code-level vulnerabilities (XSS, CSRF, and SQL injection) were addressed using design-level solutions.

The number of reported vulnerabilities decreased over time. This would indicate an increased maturity of the security of the software. The investigation of the sources of the recently reported vulnerabilities revealed that they were discovered because of the use of dynamic code analysis for PHP-based applications, which is a recent development in code analysis for PHP-based software–see [15] for details about the vulnerabilities. Thus, the security of the software is getting mature but with respect to known vulnerability types and known techniques to identify them.

Initially, we believed that code chunks that fix a given vulnerability may give us hints on the relation between code changes and introduced vulnerabilities. Thus, we selected 2 versions of Zen Cart, V1.1.3 and V1.1.4 and analyzed the SQL injection vulnerability in V1.1.3, which was mitigated in V1.1.4. We found that the vulnerability was in $includes/application.php$ file. Thus, the technique helps to identify code-based vulnerabilities fixes, such as SQL injection.

We extended our investigation; we used Meld[5] to compare Zen Cart versions Zen-Cart-V1.1.0 and Zen-Cart-V1.1.4d and to derive the differences between the two versions. As a result we identified (1) new files added to V1.1.4d version, (2) updates to existing files, and (3) files that have been removed from the new version. We were able to relate only 3 vulnerabilities out of the 35 reported. The main challenge

[5]http://www.meldmerge.org/

we had is that most of the vulnerabilities were not specific to identify related code changes. As a lesson, we believe that given the quantity of changes, it is difficult to trace manually with high confidence code changes to introduced vulnerabilities.

C. Zen Cart threat model

We used Zen Cart collaboration diagram of Figure 1 and we derived a threat model of the software using Microsoft Threat modelling tool[6]. The threat modeling tool uses the STRIDE method[7] to generate the list of threats for the software. The tool does not assume trust in the internal network. Table II lists the threats to the software that the tool generates–they are all of high priority. Table II shows a long list of abstract threats.

The threat modeling techniques, such as STRIDE, assume the architecture diagram is an abstraction of the the implemented software. In practice, the agile team use the diagrams to explain and communicate. Instantiating the diagram would not produce the developed software. For instance, Figure 1 does not abstract Zen Cart code. In fact, Zen Cart team develops a new architecture document for each version of the software that it submits for PA-DSS certification. The DFD diagram that would be used for the threat modeling does not correspond to the code of the software. Thus, the identified threats are not for the implemented software.

As a second attempt, we tried to extract the call graph of the software to identify its attack surface. (A softwares attack surface is the set of ways in which an adversary can enter the system and potentially cause damage [16].) We used PHP CallGraph[8], RIPS (RIPS is a static code analysis tool to automatically detect taint-style vulnerabilities in PHP applications.),[9] and doxygen[10] to extract the call graph of Zen Cart. In the 3 cases, the resulting call graph was very big due to large number of functions present in Zen Cart. We did not find a way, so far, to use them to identify the attack surface.

We note here that Zen Cart team members do not perform threat modeling. They use instead the OWASP Top 10 list as guideline of the vulnerabilities to address. In addition, the software implements security mechanisms that implement the PA-DSS compliance/security requirements.

D. Penetration testing of Zen Cart

We tested Zen Cart V1.5.4 manually with respect to the Open Web Application Security Project (OWASP) [17] top 10 vulnerabilities for web applications. We tested manually a set of security test cases using the Burp Suit[11]. We also performed static code analysis for Zen Cart using RIPS. We did not identify with high confidence existing vulnerabilities. For instance, the version of RIPS (V0.55 latest version) that we used was too imprecise in terms of extracting function call graph out of Zen Cart source code. Zen Cart source code includes hundreds of functions while RIPS V0.55 does not support source code that has more than one hundred functions.

IV. Secure code change at Zen Cart

We report in this section the results of an interview we performed with Mr Ian Wilson, a lead developer at Zen Cart on 25 March 2016. The interviewee is responsible for the security aspects of the software. He has about 20 years of experience in software development and more than 10 years of experience with Zen Cart. The goal of the interview was to provide a practical case of how a software organization iteratively develop secure software.

We used the common approach for interview processing. We prepared an interview protocol and interviewed our expert for about 40 minutes. We transcribed the interview and coded it. The results are below. We did not interview other experts since we expected that we would not get more responses for the questions we had.

Zen Cart is an e-commerce software. The development team uses the Open Web Application Security Project (OWASP) top 10 as a guideline for the severe threats; they do not use any threat modeling tool. The application is developed to be compliant with PA-DSS security requirements since 2007. The team uses skipfish[12] and burp suite[13] for testing the application.

Changes that affect the security of the application are of three categories: (1) security requirements changes, (2) code changes, and (3) security mechanism changes. These changes are not frequent. For instance, only one of the last 50 code commits at the date of the interview was related to security.

(Change category 1) The Security Standards Council of the Payment Card Industry (PCI) updates the PA-DSS security requirements often to consider emerging threats. For each new version of the standards, Zen Cart has to assess the compliance of the application with all the security requirements. This requires performing all the required tests and verifications. This allows the customers to comply with the new standards. Nevertheless, a compliance with the standards is valid, in average, for one year, and to the maximum for two years.

(Change category 2) The team implements often new features and fixes bugs. The team uses JIRA[14] and fisheye[15] to trace the code changes. The developers discuss the impact of each code change on the PA-DSS security requirements, especially on the authorization mechanisms and run the authorization tests. They also receive vulnerability reports from security companies that they need to address. For example, the team received in March 2016 a report about a set of XSS vulnerabilities from trustwave [15].[16] The team worked on

[6]https://www.microsoft.com/en-us/download/details.aspx?id=49168

[7]STRIDE stands for Spoofing, Tampering, Repudiation, Information disclosure, Denial of Service, and privilege Escalation.

[8]http://phpcallgraph.sourceforge.net/

[9]http://rips-scanner.sourceforge.net/

[10]http://www.doxygen.com/

[11]https://portswigger.net/burp/

[12]http://code.google.com/p/skipfish/

[13]https://portswigger.net/burp/

[14]https://www.atlassian.com/software/jira/

[15]https://www.atlassian.com/software/fisheye

[16]The company developed a new tool and used it to test Zen Cart to prove the efficacy of the tool.

individual fixes for vulnerabilities, which were not efficient. Then they decided to change the architecture of the software to use new data sanitization methods. The developers spent about one month discussing the solution for the vulnerabilities and solved them with only 200 lines of code.

(Change category 3) Sometimes, security mechanisms become insecure and require redesign. For example, initially Zen Cart stores the passwords using an empty file hashing system. The mechanism was found later to be insecure. PHP introduced in Version 5.5 a new password hashing function that replaces the previous mechanism. The team updated Zen Cart to use the new feature. However, the solution works for PHP version 5.5 and subsequent version, but not for the previous ones. Zen Cart team addressed the compatibility problem by introducing a compatibility layer that allows to use the PHP built-in function for PHP version 5.5 and subsequent versions and a function that mic-mic the built-in (but probably less performant one) for the older PHP versions.

Tests of security requirements are believed to be misleading. Sometimes, the tests that the developers design do not cover all the cases. For example, Zen Cart team performed successfully the tests for the new password hashing mechanism discussed above. When reviewing the code changes, the team discovered that some old versions of the software limit the size of the database field used to store the hashed password. This limit makes the implemented new mechanism inefficient. Thus, Zent Cart team members have limited confidence in the security tests; they support that with manual review of the code.

Zen Cart application, as many open source software, does not have architecture diagrams. These diagrams are believed to become obsolete after few code changes. The Zen Cart team writes a new architecture document for each version of the application to be submitted for PA-DSS certification. The analysis of the impacts of code changes on the security requirements of the software are therefore assessed by the developers who are familiar with the code. For them, it suffices to analyze the trace of the code changes through JIRA and fireeye, to know whether the change has an impact on the security requirements or not. This seems to be error prone.

The main lesson learned from introducing security practices to the development of Zen Cart is the value of security architecture. Zen Cart team members have spent much time identifying and fixing individual vulnerabilities, such as XSS and SQL injections. This effort could have been much less if the team focused in the early releases on designing a security architecture for the application. Over the years, the interviewee learned to never trust data that comes from external sources, such as user input, data from third party software, and environment variables. Therefore, all external data must be systematically sanitized.

V. SecureAgile: A tool for tracing the impact of code changes on software security

This section describes a prototype we are developing to identify the impacts of code changes on the security of software.

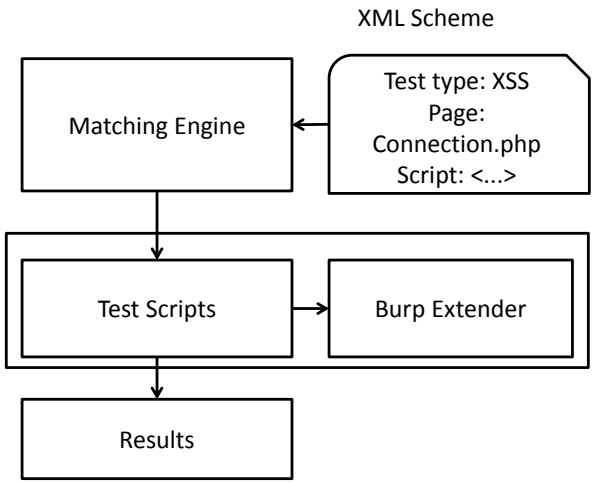

Figure 2. Architecture of Penetration testing automation tool for Zen Cart

A. Automation of penetration testing

Initially, we used Selenium[17] and JSoup[18] libraries to develop a small tool to automate few penetration tests (tests for SQL and Cross Site Scripting (XSS) injections) that we performed for Zen Cart V1.5.4 using Burp Suite. The test cases that we performed require data entries as parameters and are repetitive. Thus, we designed XML Scheme that supports specifying these test cases. Listing 1 shows examples of specifying SQL and XSS injections using the scheme. A test case includes test type (XSS or SQL Injection), web page to be tested and the script to be executed.

Listing 1. Example of test cases.

```
<test>
    <TestType>
        Cross Site Scripting
    </TestType>
    <Page>
        index.php, demoadmin/index.php
    </Page>
    <Script>
        &lt;script&gt;alert ('This is an
        XSS Vulnerability');&lt;/script
        &gt;
    </Script>
</test>

<test>
    <TestType>
        SQL injection
    </TestType>
    <Page>
        demoadmin/login.php
```

[17]http://www.seleniumhq.org/
[18]https://jsoup.org/

```
    </Page>
    <Script>
        username = admin Password =
        anything OR 'x' = 'x'
    </Script>
</test>
```

We also developed a tool, called Penetration Testing Engine (PTE), that extracts for each test case specified in the given XML file the test parameters and performs the test case using these parameters. Figure 2 shows the architecture of the penetrating testing automation tool. First, the tool reads the test cases file. For each test cases, it calls the routine that tests the vulnerability associated with the case and give it the required parameters. The routine returns true/false as a result which indicates whether the test was successful or not.

We experimented with the tool on Zen Cart V1.5.4 using 25 test cases. All the test cases failed.

B. Visual design of security assurance cases

A security assurance case [18] (known more succinctly as a security case) uses a structured set of arguments and a corresponding body of evidence to demonstrate that a system satisfies specific claims with respect to its security properties [19]. Weinstock et al. [5] presented how security assurance cases are created using graphical Goal Structuring Notation [20] which we use for our Eclipse plug-in.

A security case bears considerable resemblance to a legal case, and demonstrates that security claims about a given system are valid. Persuasive argumentation plays a major role, but the credibility of the arguments and of the security case itself ultimately rests on a foundation of evidence [19]. It allows to combine a range of complementary security assessment techniques including code review, model verification, static analysis of code-level requirements (e.g. pre- and post-conditions), static analysis of vulnerability signatures, and penetration tests [19]]. Assurance cases summarize the results of these assessments and show how the results are related to security assurance claims. The security requirements correspond to the claims and sub-claims and the results of the assessment activities correspond to the evidences.

We developed a Security Assurance Case Plug-in (SACP) as an Eclipse plug-in to design security assurance cases in the Eclipse development environment. The plug-in uses the Eclipse Modeling Framework (EMF) [21] and the Graphical Modeling Framework (GMF) [22]. Figure 3 shows a sample Security Assurance Case using the SAC plug-in.

The plug-in palette has 5 elements:

1) Goal: It is the top level security claim of the Security Assurance Case.
2) Context: A set of information related to the scope of the top-level claim.
3) Strategy: The logic for developing the arguments that support the claim. Figure 3 shows an example of strategy: a split of the claim into three sub-claims; the claim is satisfied if the three sub-claims are satisfied.
4) Solution: It corresponds to the available evidence that justifies that the related claim is satisfied.
5) Link: It connects 2 elements of the security assurance case.

Figure 3 shows a simple security assurance case we specified for Zen Cart using SACP. The main claim of the security case is that Zen Cart is acceptable secure (G1). This claim is decomposed into 4 sub-claims: secure storage of cardholder data (G2.1), sub-claim only authenticated users can access the software (G2.2), the payment application activities are logged (G2.3) and data are exchanged over secure public network (G2.4). Sub-claims G2.1, G2.3, and G2.4 are not satisfied yet and sub-claim G2.2 is satisfied.

Sub-claim G2.2 is also decomposed into 4 sub-claims. The first sub-claim, use unique user-ID (G2.2.1), is satisfied by the verification that the user-ID is generated using a secure random-number generator. The second claim, secure transmission of passwords (G2.2.2) is satisfied by the verification that passwords are sent through HTTPS connections. The thirst sub-claim, authenticate each transaction (G2.2.3), is satisfied by testing the code for CSRF vulnerability type and reviewing the code to ensure the use of hidden keys for each transaction. The last sub-claim, correct session management (G2.2.4), is satisfied by reviewing the code to ensure correct management of the sessions.

Software is aimed often to comply with security standards, including corporate security requirements. For instance, the Zen Cart team decided in 2007 to seek Payment Application Data Security Standard (PA-DSS) certification [23]. The PA-DSS certification is required by banks for software that use Credit cards for payments. As future work we will investigate the use SACP to specify the security requirements of standards, such as PA-DSS.

C. Visual mapping of user stories to the security artifacts

We developed a second Eclipse plug-in for visual mapping of user stories to security artifacts, named User Story Security Mapping Plug-in (USSMP). Figure 4 shows the specification of a sample user story using USSMP. Each user story is specified with a name, a responsible person, a set of security requirements, a set of security tests and a status–e.g., new, in-progress, and closed. The security requirements correspond to claims in the security assurance cases and the security tests correspond to evidences specified in the security case of the given software. A change of the code related to a given user story implies that the security requirements associated to the user stories are potentially affected, they need to be reassessed.

The security tests associated with user stories correspond to security tests specified in the penetrating testing XML file of the given software–see Section V-A. The PTE matches the test cases specified in the USSMP and performs the corresponding tests.

Security assurance uses several assessment techniques besides automated tests, such as code review. The plug-in should be extended to support the other assessment techniques.

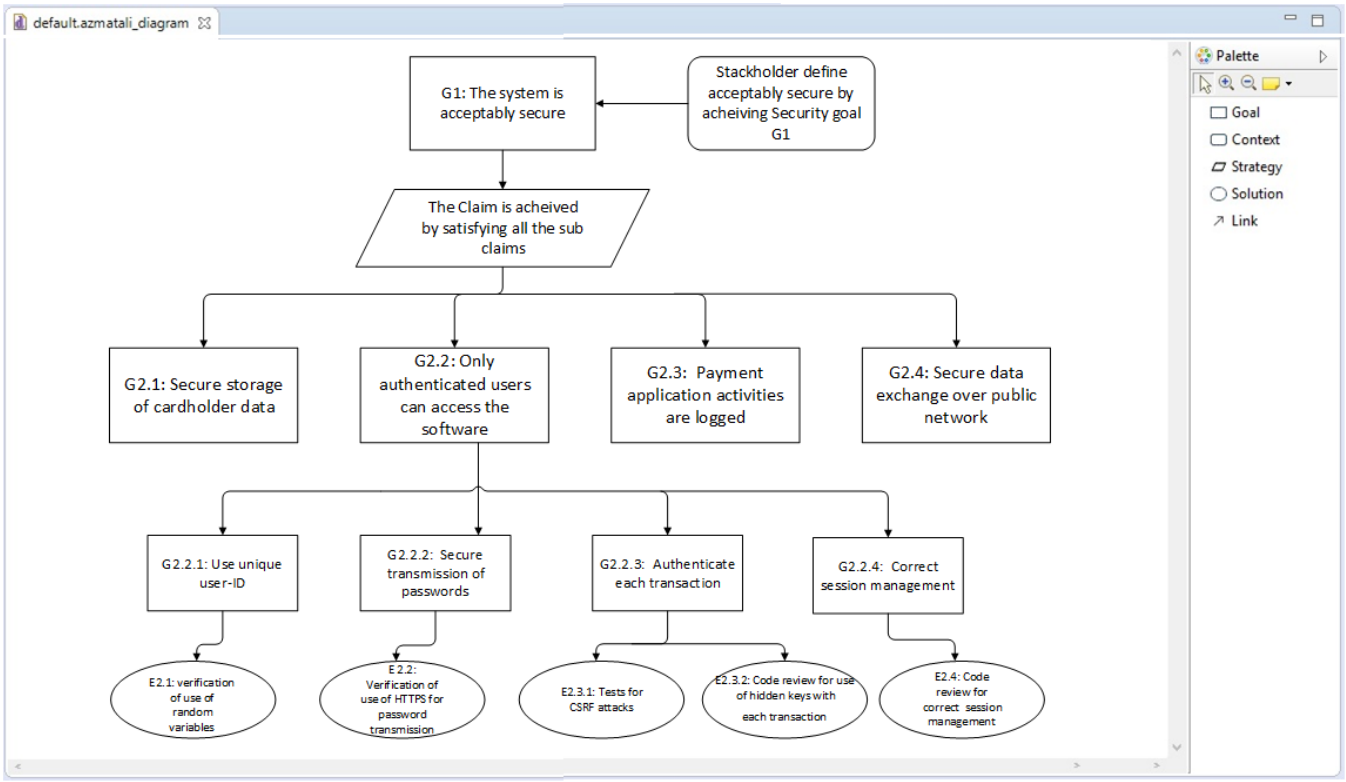

Figure 3. A partial security assurance case for Zen Cart developed using our Security Assurance Case Plug-in (SACP).

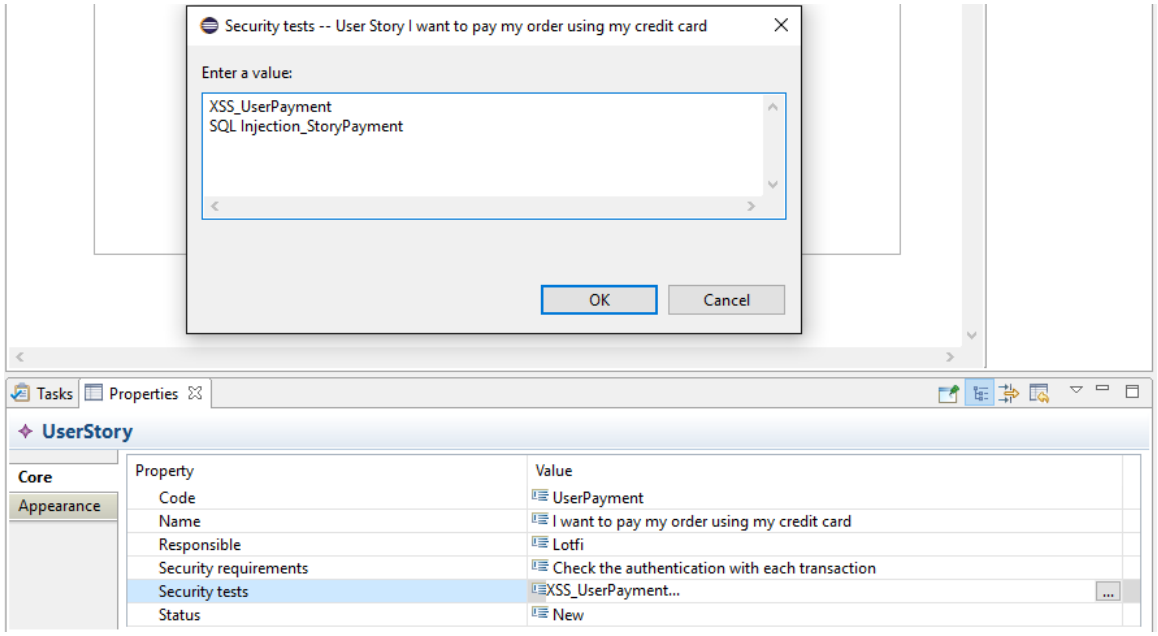

Figure 4. User Story Security Mapping Plug-in (USSMP).

VI. Conclusion

Security assurance is believed to be challenging when software is developed in successive iterations. We learned from this study that three changes impact the security of software: security requirements changes, code changes, and security mechanism changes. We also learned that changes that impact software security are rather not frequent but they are time consuming. In addition, we found that agile development team do not maintain up-to-date architecture diagram for their software, which makes threat modeling using such diagrams is useless.

This paper presents a tool we are developing to trace code changes to security assurance artifacts. We believe that the technique will help reducing the security reassessment of software

Currently, we are addressing several aspects related to the techniques we proposed. We are focusing now on automatic architecture recovery of software from the source code and automatic threat modeling of web applications. The results of this research would allow extracting threat models of software rather than deriving a list of threats from an obsolete representation of the given software.

Acknowledgment

The authors thank Ian Wilson from Zen Cart for the interview and information. They also thank Pritam Dash for performing early set of security tests on Zen Cart.

References

[1] J. Jürjens, L. Marchal, M. Ochoa, and H. Schmidt, "Incremental security verification for evolving umlsec models," in *Proc. of the 7th European Conference on Modelling Foundations and Applications*, ECMFA'11, (Birmingham, UK), pp. 52–68, 2011.

[2] H. Oueslati, M. M. Rahman, L. B. Othmane, I. Ghani, and A. F. B. Arbain, "Evaluation of the challenges of developing secure software using the agile approach," *International Journal of Secure Software Engineering (IJSSE)*, vol. 7, 1 2016.

[3] L. Ben Othmane, P. Angin, H. Weffers, and B. Bhargava, "Extending the agile development process to develop acceptably secure software," *IEEE Transactions on Dependable and Secure Computing*, vol. 11, no. 6, pp. 497–509, 2014.

[4] L. Ben Othmane, P. Angin, and B. Bhargava, "Using assurance cases to develop iteratively security features using scrum," in *Proc. of Ninth International Conference onAvailability, Reliability and Security (ARES), 2014*, (Fribourg, Switzerland), pp. 490–497, 2014.

[5] C. B. Weinstock, H. F. Lipson, and J. Goodenough, "Arguing security - creating security assurance cases," 2007.

[6] Z. C. Team, "Zen cart - implementation guide." Available at https://www.zen-cart.com/docs/implementation-guide-v153.pdf. Accessed on October. 2015.

[7] R. S. Arnold, *Software Change Impact Analysis*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1996.

[8] S. Lehnert, "A taxonomy for software change impact analysis," in *Proc. of the 12th International Workshop on Principles of Software Evolution and the 7th Annual ERCIM Workshop on Software Evolution*, IWPSE-EVOL '11, (Szeged, Hungary), pp. 41–50, Sep. 2011.

[9] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley, "Chianti: A tool for change impact analysis of java programs," in *Proc. of the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, (Vancouver, BC, Canada), pp. 432–448, Oct. 2004.

[10] S. A. Bohner, "Extending software change impact analysis into cots components," in *Proc. of the 27th Annual NASA Goddard/IEEE Software Engineering Workshop*, (Greenbelt, MD), pp. 175–182, 2002.

[11] "Security engineering for lifelong evolvable systems." Available at http://www.securechange.eu/content/project-overview. Accessed on Apr. 2016.

[12] W. Raschke, M. Zilli, P. Baumgartner, J. Loinig, C. Steger, and C. Kreiner, "Supporting evolving security models for an agile security evaluation," in *Proc. of the IEEE 1st Workshop on Evolving Security and Privacy Requirements Engineering (ESPRE)*, (Karlskrona, Sweden), pp. 31–36, Aug. 2014.

[13] "Zen cart the art of e-commerce." Available at http://www.zen-cart.com/. Accessed on Jan. 2016.

[14] A. Ramirez, P. Vanpeperstraete, A. Rueckert, K. Odutola, J. Bennett, L. Tolke, and M. van der Wulp, "Argouml user manual a tutorial and reference description." Available at https://www.tjhsst.edu/~rlatimer/uml/argomanual-0.18.1.pdf,, 2003. Tech. rep., 2000–2009.

[15] Trustwave, "Twsl2016-006: Multiple xss vulnerabilities reported for zen cart." Available at https://www.trustwave.com/Resources/SpiderLabs-Blog/TWSL2016-006--Multiple-XSS-Vulnerabilities-reported-for-Zen-Cart/. March 25, 2016.

[16] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Transactions on Software Engineering*, vol. 37, pp. 371–386, May 2011.

[17] "The open web application security project OWASP." Available at https://www.owasp.org/. Accessed on Jan. 2016.

[18] J. L. Vivas, I. Agudo, and J. López, "A methodology for security assurance-driven system development," *Requirements Engineering*, vol. 16, no. 1, pp. 55–73, 2010.

[19] H. Lipson and C. Weinstock, "Evidence of assurance: Laying the foundation for a credible security case." Available at https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/assurance/973-BSI.htm, May 2008.

[20] T. Kelly and R. Weaver, "The goal structuring notation a safety argument notation," *Proc. Dependable Systems and Networks - Workshop on Assurance Cases*, July 2004.

[21] F. Budinsky, D. Steinberg, E. Merks, R. Ellersick, and T. J. Grose, *Eclipse Modeling Framework: A Developer's Guide*. Addison Wesley, 2003.

[22] "Graphical modeling framework documentation." Available at http://wiki.eclipse.org/Graphical_Modeling_Framework/Documentation/Index#GMF-Tooling_.2F_Generation_Framework. Accessed on 4th Nov. 2016.

[23] "Payment application data security standard–requirements and security assessment procedures." Available at https://www.pcisecuritystandards.org/minisite/en/docs/PA-DSS_v3.pf, Nov. 2013.