A threat pattern for the "Cross-Site Scripting (XSS)" attack

ROHINI SULATYCKI, Florida Atlantic University
EDUARDO B. FERNANDEZ, Florida Atlantic University

We present a threat pattern that describes cross-site scripting (XSS) attacks. Cross-Site Scripting is listed as number three risk on the 2013 OWASP Top 10 list; it is an attack made possible due to the lack of user input validation or output escaping, which allows attackers to inject malicious code. The pattern describes how the attack is performed, which vulnerabilities it exploits, and how to stop it.

Categories and Subject Descriptors: D.2.11 **[Software Engineering]** Software Architecture – Patterns; D.5.1 D.4.6 **[Security]**

General Terms: Design

Key Words and Phrases: Threat patterns, XSS, security patterns, application security, vulnerabilities, exploits, sensitive data

1. INTRODUCTION

To design a secure application, we first need to understand possible threats to the application. For this purpose we introduced the concept of misuse patterns (Fernandez et al., 2007), which describe how an information misuse is performed. Later, we proposed *threat patterns* (Uzunov and Fernandez, 2013), to describe the steps of an attack leading to several related misuses. Both patterns describe how an attack is performed from the point of view of the attacker. They define the environment where the attack is performed, countermeasures to stop it, and provide forensic information in order to trace the attack once it happens. For example, a security defense misconfiguration is a *vulnerability,* taking advantage of this vulnerability is a threat (potential attack) which can lead to reading unauthorized information (a *misuse*). In particular, threat patterns are useful for developers because once they determine that a possible attack can happen in the environment, the pattern will indicate what security mechanisms are needed as countermeasures. Also, threat patterns can be very useful for forensic examiners to find useful evidence information after the attack has been performed. Finally, they can be used to evaluate an existing system and verify if it can handle specific threats. Note that a threat pattern describes a complete attack, e.g. stealing information from a database, not just specific steps used to perform the attack, such as SQL injection or buffer overflow (both can be used in the same threat). Threat patterns take advantage of specific vulnerabilities and can be described with respect to the corresponding vulnerability or they can be defined with respect to a set of vulnerabilities that allow the attack to proceed.

The Open Web Application Security Project (OWASP) publishes a list of the ten most critical web application security risks called the OWASP Top 10 (OWA10). The OWASP Top 10 is a widely adopted document for application security and has been developed with broad consensus from security experts worldwide. However, their descriptions of common threats do not emphasize architectural aspects and they mix vulnerabilities with threats. We are converting the OWASP descriptions into threat patterns because we believe they will be more useful in this way; we have already written patterns for four of these threats: "Compromising applications using components with known vulnerabilities" and "Direct access to objects using uncontrolled references" (Sulatycki and Fernandez 2015a); "Security Misconfiguration" and "Sensitive Data Exposure" (Sulatycki and Fernandez 2015b).

Cross-Site Scripting has been one of the top 10 security issues for web applications for several years and is considered one of the most prevalent web application security threats (OWA10). Cross-site scripting attacks have surpassed buffer overflows as the world's most commonly reported security threat (Bates et al.). XSS is a

Author's address: R. Sulatycki, Dun &Bradstreet, 3 Sylvan Way, Parsippany, New Jersey 07054; email: sulatyckir@dnb.com; E.B.Fernandez, Department of Computer and Electrical Engineering and Computer Science. Florida Atlantic University 777 Glades Rd. Boca Raton, FL 33431; email: fernande@fau.edu

type of injection attack, made possible due to the lack of user input validation (Shar and Tan 2012), or output escaping.

We present here a threat pattern to describe XSS attacks. Our audience includes architects, system designers, and web application developers. Section 2 presents the template we use to describe threat patterns (Fernandez, 2013); it is based on the POSA template (Buschmann et al., 1996) with tailoring of some sections. In Section 3, we present a threat pattern for XSS. Section 4 presents some conclusions and possible future work.

2. TEMPLATE FOR THREAT PATTERNS

2.1 Name

The name of the pattern should correspond to the generic name given to the specific type of threat in standard attack repositories.

2.2 Intent or thumbnail description

A short description of the intended purpose of the pattern (what problem it solves for an attacker).

2.3 Context

It describes the generic environment including the conditions under which the attack may occur. This may include minimal defenses present in the system as well as standard vulnerabilities of the system.

2.4 Problem for the attacker

From an attacker's perspective, the problem is how to find a way to attack the system. The *forces* indicate what factors may be required in order to accomplish the attack and in what way; for example, which vulnerabilities can be exploited.

2.5 Solution

This section describes the solution of the attacker's problem, i.e., how the attack can reach its objectives and the expected results of the attack. UML class diagrams show the system units involved in the attack. Sequence or collaboration diagrams show the exchange of messages needed to accomplish the attack.

2.6 Affected system components (Where to look for evidence)

The pattern should represent, typically using a class diagram, all components that are important to prevent the attack. From a forensic viewpoint, it describes what information can be obtained at each stage tracing back the attack and what can be deduced from this data.

2.7 Known uses

Specific incidents where this attack occurred are preferred but for new vulnerabilities, where an attack has not yet occurred, specific scenarios for the potential attack are enough.

2.8 Consequences for the attacker

Discusses the benefits and drawbacks of a misuse pattern from the attacker's viewpoint. The enumeration includes good and bad aspects and should match the forces.

2.9 Countermeasures

It describes the security measures necessary in order to stop, mitigate, or trace this type of attack. This implies an enumeration of which security patterns are effective against this attack.

2.10 Related Patterns

Discusses other threat patterns with different objectives but performed in a similar way or with similar objectives but performed in a different way.

3. CROSS-SITE SCRIPTING

3.1 Intent

In a Cross-Site Scripting attack attackers write scripts in web applications that will lead to attacks to a target web application. The main idea is to use special characters to make the browser interpreter switch from a data context to code execution that will perform a misuse in the victim's application.

3.2 Context

Web applications that include user supplied input and outputs to other applications. These applications typically use HTML, Javascript, and Ajax.

3.3 Problem for the attacker

From the attacker's perspective, the problem is getting to execute scripts in a victim's browser.

These attacks can be performed by taking advantage of the following vulnerabilities:

- Any text based script can be used to exploit the interpreter in the browser. That means the attacker has many possible web sites to attack.
- Unrestricted user input. Many web sites need to interact with users and usually do not control this input.
- The application often has output functions that can write to other applications.
- Technologies such as Ajax, allow complex interactions between the users and the server, which can be exploited to perform attacks and avoid detection.

The attack is facilitated by:

- There are a number of dictionaries such as xssed.org that publish XSS-related vulnerabilities in specific products.
- The existence of bug bounty programs provides an economic incentive to search for and to disclose information on vulnerabilities. Additionally, vulnerability disclosure is seen as a status symbol for many security experts.
- The Browser Exploitation Framework (BeEF) can be used to "hook" one or more web browsers and use them as pivot points for launching directed command modules and further attacks against the system from within the browser context.

3.4 Solution

The attack exploits an application's HTML output function that references user input and sends data to the browser (Shar and Tan 2012). The attacker introduces special characters that make the browser switch from a data to a code context so that script can be activated to produce output.

3.5 Variants

There are two types of cross-site scripting:

- *Stored (persistent) cross-site scripting* - malicious payloads are made persistent in a permanent store such as a database and the attack affects any user of the web site who visits the infected page.
- *Reflected cross-site scripting* – The victim is tricked on clicking on a malicious link, which is executed in the context of the victim's browser.

3.5.1 Structure (Affected system components)

Figure 1 shows a class diagram for compromising applications with XSS vulnerabilities. The *Attacker* submits input that includes a malicious script. The *Application* renders the input back and the *Browser* is tricked into executing the malicious script. The execution of this script will attack the *Victim* who is a user of the Application.

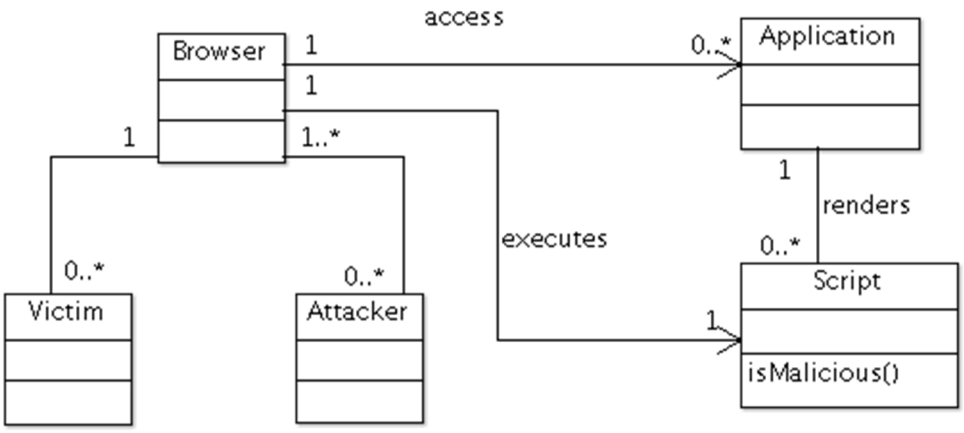

Figure 1. Class Diagram for Use Case "Compromising Applications with XSS attack"

3.5.2 Dynamics

The following use cases describe the sequences of two typical attacks.

UC1: Stored XSS attack (Fig. 2)
Summary: The Attacker compromises an application vulnerable to stored XSS.

Actor: Attacker

Precondition: The Attacker must be able to submit input into the application.

Description:

- The attacker submits a text-based script to the application.
- The application stores the script.
- The victim visits an infected page of the application.
- The application executes the script in the victim's browser.
- The attacker then compromises the victim's application.

Postcondition:
The victim, who is an application user, is compromised: XSS can be used to steal sensitive information such as usernames and passwords, perform session hijacking, remotely control or monitor the user's browser, poison cookies, impersonate a web page used to gather information, including credit card numbers or used as a pivoting point for other attacks.

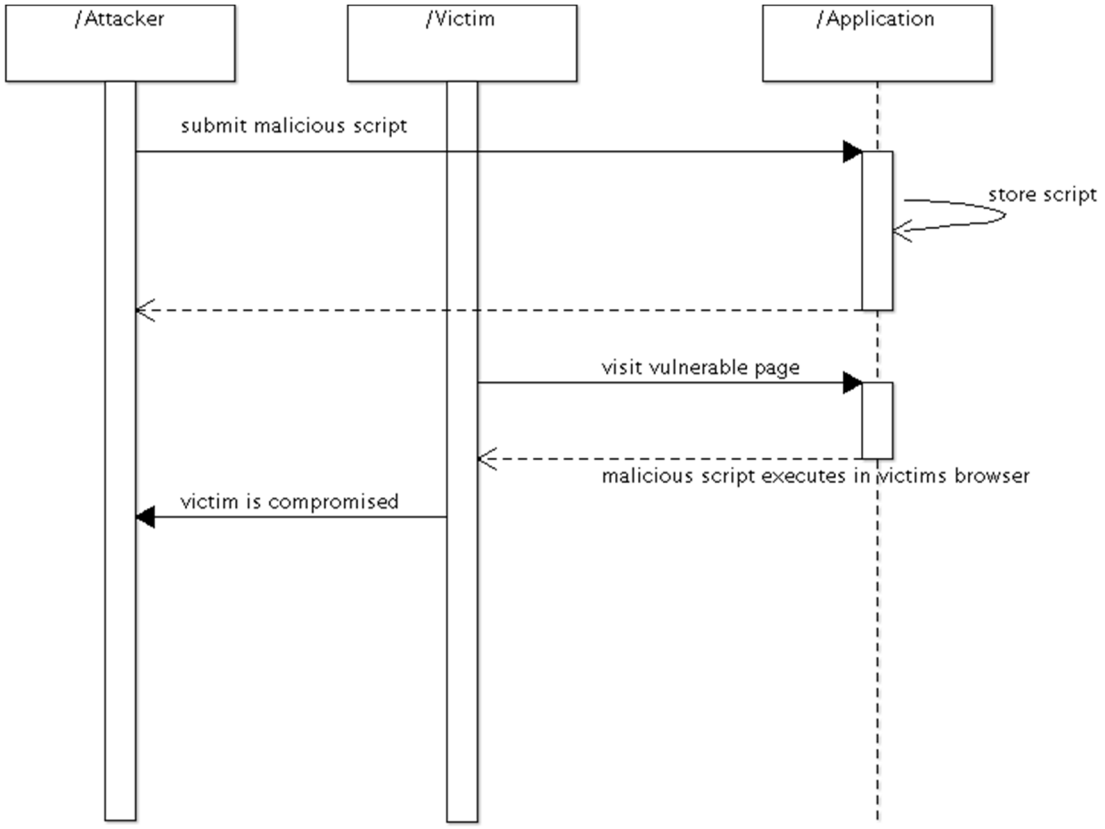

Figure 2. Sequence Diagram for the use case *Compromising an application with stored XSS*

UC2: Reflected XSS attack (Fig. 3)
Summary: The Attacker compromises an application vulnerable to reflected XSS.

Actor: Attacker

Precondition: The Attacker must have access to the application.

Description:

- The attacker creates a link containing malicious script targeting the vulnerable application and makes it available to the attacker e.g. by sending an email to the victim containing the link
- The victim clicks on the malicious link.
- The application executes the script in the victims' application.
- The attacker then compromises the victim.

Postcondition:
The victim is compromised, see possible benefits for the attacker in Section 3.8.

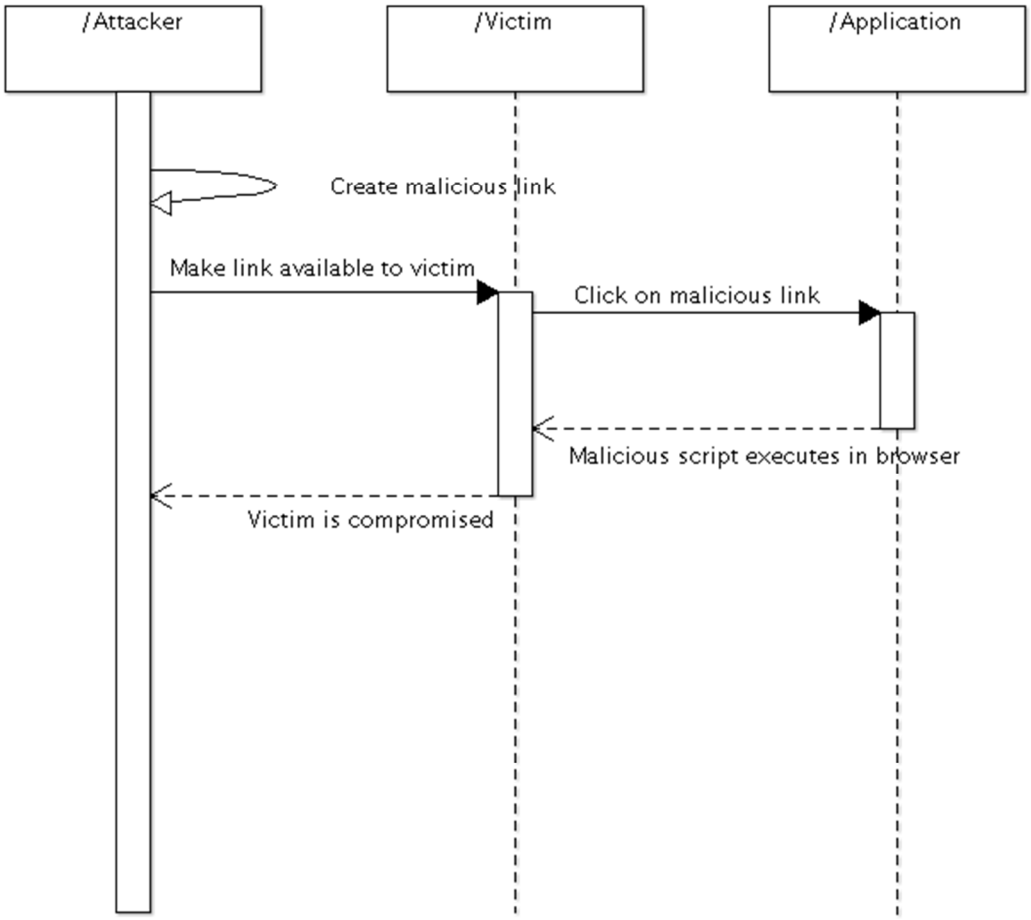

Figure 2. Sequence Diagram for the use case *Compromising an application with reflected XSS*

3.6 Affected system components (Where can we find evidence of this attack?)

- We can audit a compromised application and look for evidence of XSS attacks in application logs or persistent data store.

3.7 Known uses (incidents)

Some incidents where these patterns (apparently) were used are:

- In April 2015, another critical XSS vulnerability was discovered that could allow commenters to compromise a WordPress site. (WSR)
- In 2014, an XSS attack in eBay allowed attackers to redirect users to a phishing page (Infosecurity)
- In 2013, an XSS attack in Yahoo Mail allowed attackers to compromise user accounts (TNW)
- A Facebook XSS attack was misused for automatic wall posting in 2011 (Net Security).
- An XSS attack in Twitter was used to steal user cookies to hijack sessions in 2010. (Securelist, 2010)

- In October 2005, the “Samy worm” became the first major worm to use XSS for infection propagation (Whitehat)

3.8 Consequences

Some of the benefits of this threat pattern for the attacker are the following:

- An attacker can get control of applications to obtain confidential information or perform one of the misuses listed in Use Case 1 (steal sensitive information such as usernames and passwords, perform session hijacking, remotely control or monitor the user's browser, poison cookies, impersonate a web page to gather information later, or create a pivoting point for other attacks.

Possible sources of failure for the attacker include:

- The countermeasures below (section 3.9) would cause the attacker to fail.

3.9 Countermeasures

XSS can be stopped by the following countermeasures:

- Validate all user input and reject any characters that are not explicitly allowed (i.e., a white-list). This needs to be done on the server and the client. Reject also any parameters which are not of the right type, length, or range (Cobb 2009).
- In many cases, XSS is used to steal session cookies in order to hijack sessions. Mark session and other sensitive cookies as "Secure" and "HTTP-Only”.
- Perform output encoding of all data before using the data (stored or user-supplied) to generate web page content. This is particularly important when the original source of data is beyond the control of the application. (OWASP XSS) There are a number of libraries, e.g., OWASP AntiSamy that can be used for this purpose. As a rule it is considered a best-practice to utilize existing libraries which have been vetted from a security perspective rather than writing custom code.
- Consider using Content Security Policy (CSP). This policy instructs the client browser on the location and type of resources that are allowed to be loaded. Use of “Crossing boundaries” policies are also useful (Cobb 2009).
- Require scans and security testing of the entire application stack. This will reveal locations vulnerable to XSS that can then be remediated.
- Design applications using appropriate security methodologies (Uzunov et al. 2012) as a general best practice.

4. DISCUSSION AND CONCLUSIONS

Designers need to first understand possible threats before designing secure systems. However, identifying threats is not enough; we need to understand how a whole misuse is performed by taking advantage of vulnerabilities. Threat and misuse patterns appear to be a good tool to understand how misuses are performed. It is possible to build a relatively complete catalog of threat and misuse patterns for application security. Having such a catalog we can analyze a specific application and evaluate its degree of resistance to these misuses and we are in the process of building this catalog. The architecture (existing or under construction) must have a way to prevent or at least mitigate all the threats that apply to it. When potential customers use an application they must have assurance on what threat/misuses the application is able to prevent. Many providers do not want to show their security architectures; showing their list of threat/misuse patterns would give them a way to prove a degree of resistance to misuses without having to show their security details.

We illustrated our ideas with a specific pattern. We are continuing developing misuse patterns for application security in order to create a relatively complete catalog for it that can be used by application developers. Finally, we intend to incorporate these patterns into a secure systems design methodology.

ACKNOWLEDGMENTS

We thank our shepherd David Kane for his insightful comments that significantly improved this paper. He even helped improve the pattern template.

REFERENCES

D. Bates, A. Barth, and C. Jackson. Regular expressions considered harmful in clientside XSS filters. Procs. 19th International Conference on World Wide Web, WWW'10, 91–100. ACM, 2010.

BeEF, The Browser Exploitation Framework, http://beefproject.com/.

F. Braz, E.B.Fernandez, and M. VanHilst, "Eliciting security requirements through misuse activities" Procs. of the 2nd Int. Workshop on Secure Systems Methodologies using Patterns (SPattern'07). 2008.

M. Cobb, "Cross-site scripting explained: How to prevent attacks", SearchSecurity.co.UK, 11.18, 2009.

E.B. Fernandez, J.C. Pelaez, and M.M. Larrondo-Petrie, "Attack patterns: A new forensic and design tool", Procs. of the Third Annual IFIP WG 11.9 Int. Conf. on Digital Forensics, Orlando, FL, Jan. 29-31, 2007.
Chapter 24 in Advances in Digital Forensics III, P. Craiger and S. Shenoi (Eds.), Springer/IFIP, 2007, 345-357.

E.B. Fernandez, N. Yoshioka, and H. Washizaki, "Modeling misuse patterns", 4th Int. Workshop on Dependability Aspects of Data Warehousing and Mining Applications (DAWAM 2009), in conjunction with the 4th Int.Conf. on Availability, Reliability, and Security (ARES 2009). March 16-19, 2009, Fukuoka, Japan

E.B.Fernandez, "Security patterns in practice: Building secure architectures using software patterns". Wiley Series on Software Design Patterns. 2013

Infosecurity Magazine, http://www.infosecurity-magazine.com/news/ebay-under-fire-after-cross-site/

Net Security, http://www.net-security.org/secworld.php?id=10814

OWASP, "The Ten Most Critical Web Application Security Risks." 2013 https://www.owasp.org/index.php/Top_10_2013

OWASP XSS (Cross Site Scripting) Prevention Cheat Sheet, 2015,
https://www.owasp.org/index.php/XSS_%28Cross_Site_Scripting%29_Prevention_Cheat_Sheet

L.K. Shar, and H.B.K.Tan, "Defending against Cross Site Scripting Attacks," IEEE Computer 45(3), pp 55-62, 2012.

R. Sulatycki and E.B. Fernandez, 2015a, "Two threat patterns: "Compromising applications using components with known vulnerabilities" and "Direct access to objects using uncontrolled references", Procs. of 4th AsianPLoP (Pattern Languages of Programs) 2015, Tokyo, Japan, March 2015.

R. Sulatycki and E.B. Fernandez, 2015b,"Two threat patterns that exploit "Security misconfiguration" and "Sensitive data exposure", Procs. of 20th EuroPLoP (Pattern Languages of Programs) 2015, Irsee, Germany, July 2015.

Securelist, https://securelist.com/blog/incidents/29742/twitter-xss-in-the-wild/

TNW, http://thenextweb.com/insider/2013/01/31/yahoo-mail-users-still-seeing-accounts-hacked-via-xss-exploit-amid-reports-yahoo-failed-to-fix-old-flaw/

A. V.Uzunov and E.B.Fernandez, "An Extensible Pattern-based Library and Taxonomy of Security Threats for Distributed Systems"- Special Issue on Security in Information Systems of the *Journal of Computer Standards & Interfaces*. 2013. http://dx.doi.org/10.1016/j.csi.2013.12.008

Whitehat, https://www.whitehatsec.com/assets/WP5CSS0607.pdf

Wordpress Security Release, https://wordpress.org/news/2015/04/wordpress-4-2-1/