Two threat patterns that exploit "Compromising applications using components with known vulnerabilities" and "Direct access to objects using uncontrolled references" vulnerabilities

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

We present threat patterns that describe attacks against applications that take advantage of external components with known vulnerabilities or provide a direct object reference to an internal object and fail to control access to the object. These vulnerabilities might be introduced maliciously by an attacker or unknowingly by a developer and discovered later by an attacker. These patterns provide insight on how to build secure applications that use external components and are also helpful in evaluating the security of existing applications.

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: Misuse patterns, security patterns, application security, vulnerabilities, exploits, direct object reference (DOR)

1. INTRODUCTION

The Internet has been developing very rapidly during the last decade. New technologies are being introduced at a rapid pace and applications are being built using new and emerging technologies. A key feature of application development is that that most applications are composed of one or more external components, i.e. components that have been developed by other teams. Some components might be open source software (OSS) and can be incorporated into an application without any cost. Other components can be purchased from software vendors or outsourced.

There are several advantages to using external components. Incorporating external components in an application can significantly help reduce the development lifecycle. However, it also brings new security problems.

Additionally, applications available on the Internet are accessible to attackers and any flaws in the application can then be misused by an attacker to cause damage to an organization and its customers. Dynamic applications often reference internal objects using identifiers or names. For example, it is simple for a developer to use database identifiers as the object reference when accessing a database object. It is also convenient for a developer to use file names to reference a file in a parameter. However, when these object references are available to an attacker and the application does not restrict access to these objects, the attacker is then able to access and misuse unauthorized data. These attacks are in most cases not detected by automated scanners and require manual testing. (Ghafari et al. 2012)

Threat patterns combine the concept of threat libraries and taxonomies. Threat patterns provide an abstract pattern based threat taxonomy. Each threat is encapsulated in a threat pattern. Additionally, threat patterns provide a common vocabulary for software designers and implementers. These patterns reflect all the steps leading to misuses so several misuse patterns may be derived from one threat pattern. Threat patterns take advantage of specific vulnerabilities and can be described with respect to the corresponding vulnerability. We present two threat patterns that take advantage of: using components with known vulnerabilities and direct access to objects using uncontrolled references.

Section 2 presents a template used to describe misuse/threat patterns (Fernandez2013). In Section 3, we present a threat pattern for application development, Compromising Applications using Components. In Section 4 we present another threat pattern for application development, Insecure Direct Object Reference (IDOR). In

Section 5, we present some discussion on how threat patterns can be used, and in Section 6 we offer some conclusions and possible future work.

2. TEMPLATE FOR MISUSE/ THREAT PATTERNS

2.1 Name

The name of the pattern should correspond to the generic name given to the specific type of misuse 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 typical vulnerabilities of the system.

2.4 Problem

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 under 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 all components that are important to prevent the attack and are essential to the forensic examination.

2.7 Known uses

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

2.8 Consequences

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 and Forensics

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. 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 in order to identify this specific attack.

2.10 Related Patterns

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

3. COMPROMISING APPLICATIONS USING COMPONENTS WITH KNOWN VULNERABILITIES

3.1 Intent

Most applications are constructed using a combination of custom code and external components (open-source or closed-source). In many cases, the development team is not even aware of the all the components in use due to component dependencies. An attacker may take advantage of a known vulnerability or discover a vulnerability in a component to compromise the application. The compromise runs the gamut of the remaining OWASP Top 10 ranging from remote code execution to injection (OWA10).

3.2 Context

There are a number of open and closed-source components that are used by applications. These components can be created by anyone and published on the Internet. Additionally, component development can be outsourced or components can be purchased from third parties.

3.3 Problem

To perform some types of misuse it is necessary to identify an application using a component with a vulnerability known to the attacker.
The attack can be performed by taking advantage of the following vulnerabilities:

- There are a number of dictionaries that publish vulnerabilities in components.
- It is possible to scan applications for component versions and identify the vulnerable components they use.
- The emergence of vulnerability markets (Böhme, R. 2005) provides an economic incentive for researchers to search for and to disclose information on vulnerabilities. Additionally, vulnerability disclosure is seen as a status symbol for many security researchers.
- Often when vulnerability is announced, exploit code becomes publicly available on the Internet or can be purchased from underground channels.
- An attacker can create or purchase a malicious component and distribute it freely on the internet. (Crier et al. 2012)

3.4 Solution

When a user publishes a component on the Internet, anybody is able to use the component. In some cases the user will have to pay to use the component. This component might contain vulnerabilities that become publicly known. Additionally, exploit code for this vulnerability becomes publicly available. The attacker can now use this exploit code to compromise an application using the vulnerable component.

3.4.1 Structure

Figure 1 shows a class diagram for compromising application using vulnerable components.

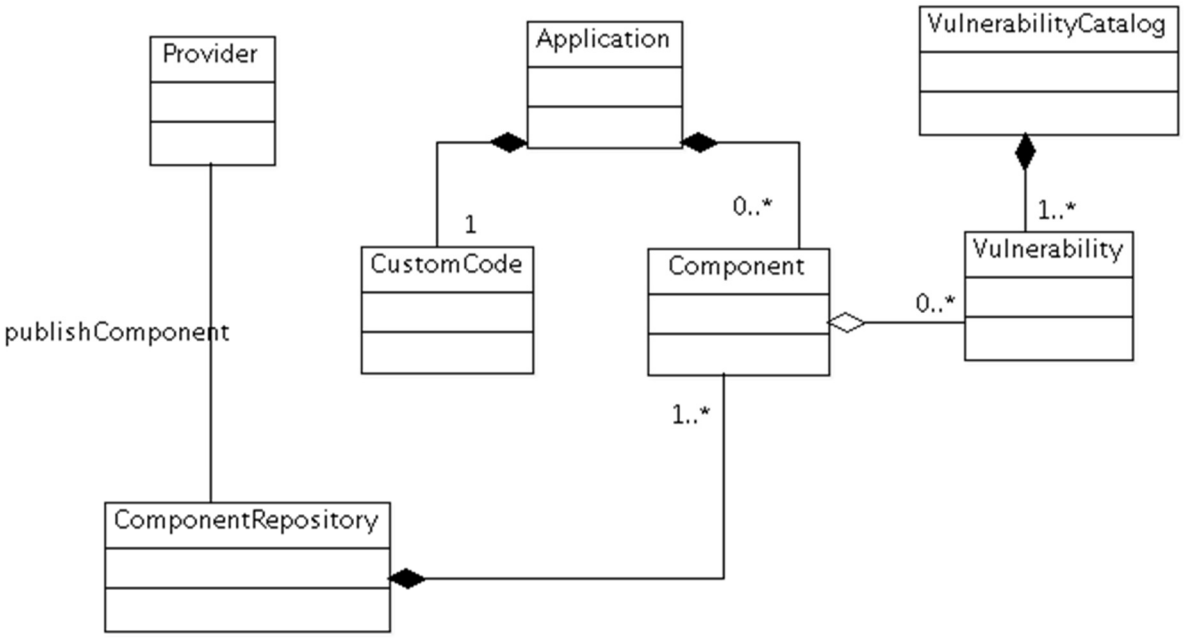

Figure 1. Class Diagram for Compromising Applications using Components with known Vulnerabilities Threat Pattern

3.4.2 Dynamics

UC1: Publish a Malicious Component (Fig. 2)
Summary: The attacker publishes a malicious component.

Actor: Attacker

Precondition: The attacker must have an account with the Provider

Description:

- The Attacker creates or purchases a malicious component from an underground channel.
- The Attacker may also modify an existing component previously published by someone else.
- The Attacker requests the Provider to upload the malicious component.
- The Provider checks if the attacker (legal user) has an account.
- The Provider uploads the component into the repository.
- The Provider sends an acknowledgement to the attacker.

Postcondition:
A vulnerable component is created and placed into the Provider's repository, so any other user can use it and get compromised

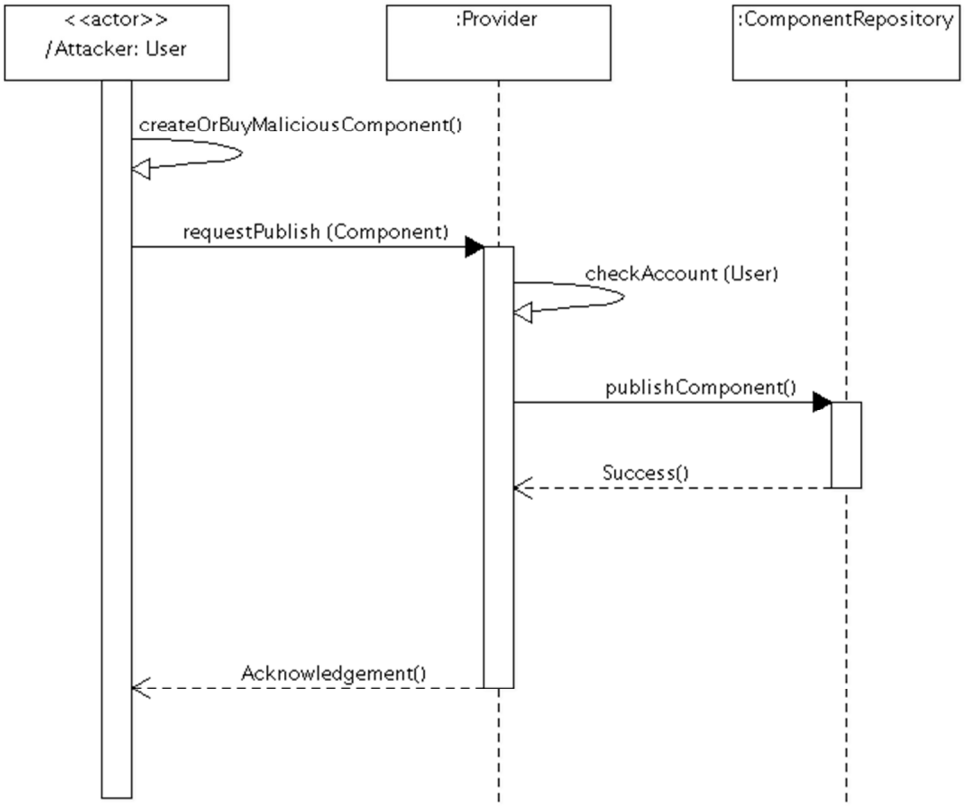

Figure 2. Sequence Diagram for the use case Publish a Malicious Component

UC2: Unknowingly Publish a Vulnerable Component (Fig. 3)

Summary: A developer publishes a vulnerable component unknowingly.

Actor: Developer

Precondition: The Developer must have an account with the Provider

Description:

- The Developer creates a component.
- The Developer requests the Provider to upload the component.
- The Provider checks if the developer (legal user) has an account.
- The Provider uploads the component into the repository.
- The Provider sends an acknowledgement to the developer.
- A Security Researcher obtains the component from the repository.
- The Security Researcher researches the component for vulnerabilities.
- The Security Researcher publishes any found vulnerabilities to the vulnerability catalog.

Postcondition:
A vulnerable component is created and placed into the Provider's repository. A security researcher publishes component vulnerabilities to the vulnerability catalog.

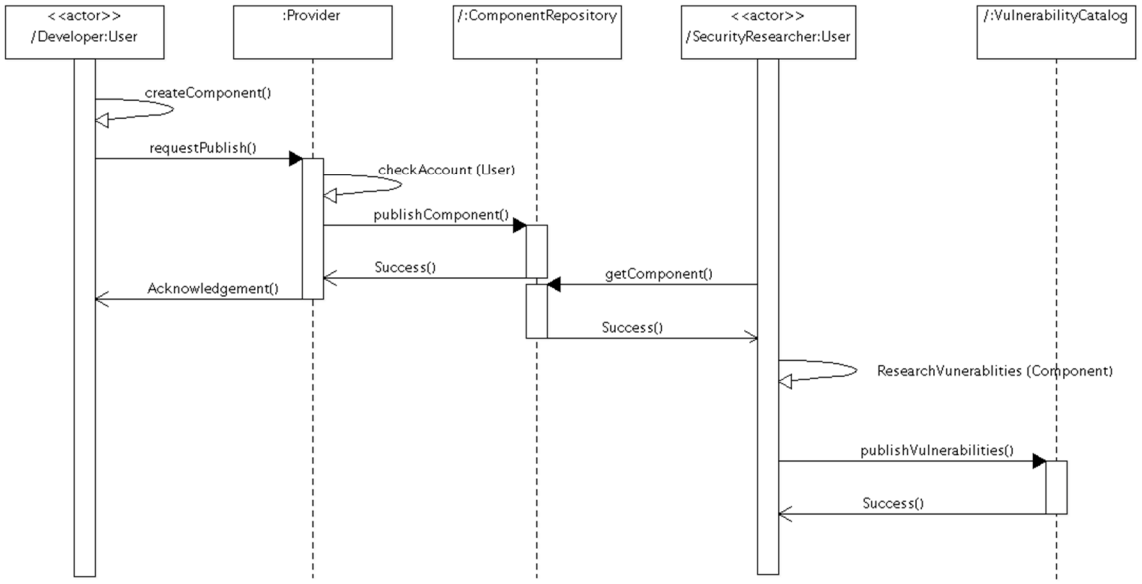

Figure 3. Sequence Diagram for the use case Unknowingly Publish a Vulnerable Component

UC3: Compromise an application using vulnerable components (Fig. 4)

Summary: An application using a vulnerable component gets compromised.

Actor: User, Attacker

Precondition: The user utilizes a vulnerable component in the application. This could be a malicious component uploaded by the attacker or a component that has known vulnerabilities. The application must be available to the attacker.

Description:

- The User requests the Provider for a component.
- The Provider checks if the user has an account.
- If the user is valid, the Provider sends the list of components
- The user selects the vulnerable component.
- The user then utilizes the component in an application and launches the application.
- The attacker scans the application and detects the vulnerable component
- The attacker either creates exploit code or finds exploit code for the vulnerability.
- The attacker then compromises the application using the exploit code.

Postcondition:
The users application is compromised.

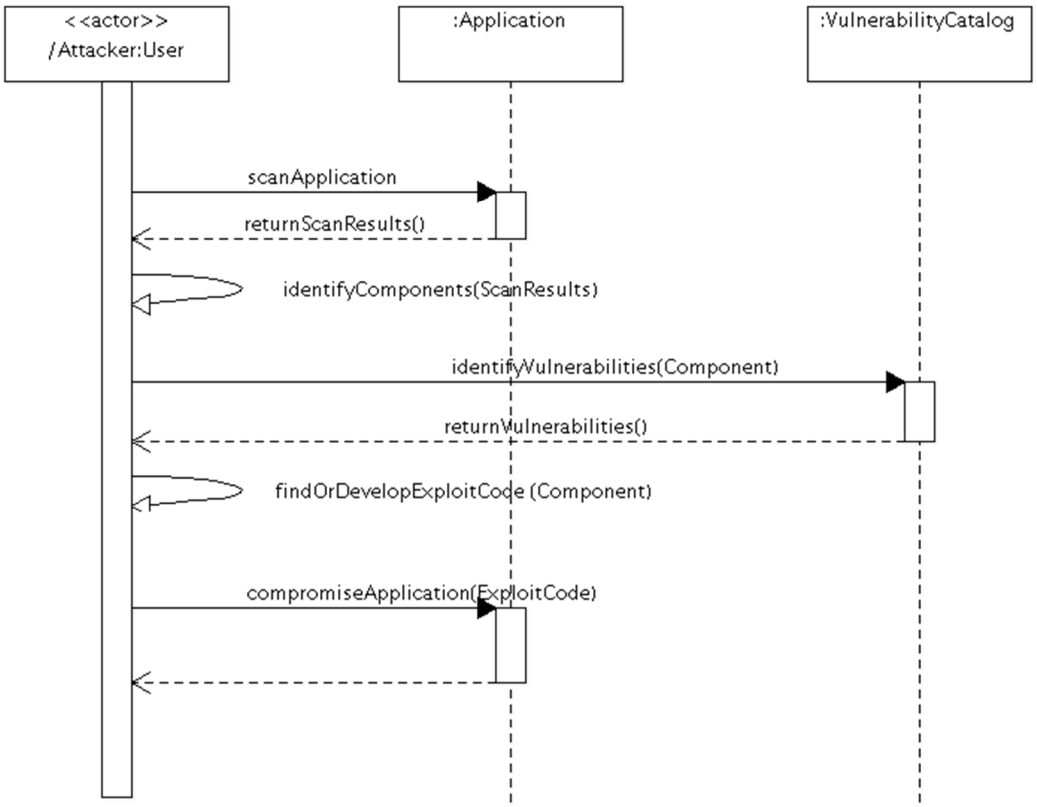

Figure 3. Sequence Diagram for the use case Compromise an Application using Vulnerable Components

3.5 Consequences

Some of the benefits of the misuse pattern are the following:

- An attacker can open an account and register malicious components into the provider's repository.
- Open source repositories that contain components can be a way to distribute malware.
- An attacker can get control of the infected system to obtain some confidential information.
- The attacker might be able to expand the attack to other systems and obtain more confidential information.

Possible sources of failure include:

- The application may have defenses against the intended attacks.
- The application may not be available for the attacker to exploit.
- The exploit code may not work.
- The infected system might be sufficiently firewalled to prevent the attacker from accessing other systems.

3.6 Countermeasures

Compromising application using components can be stopped by the following countermeasures:

- If we have a component management system that controls access to components, track the origin of components, and provide scanners that detect and repair security violations. However, if there are many components it might not be possible to perform security testing of all components.
- Verify the background of the user uploading a component; however, this is very hard to do. It might be possible to use a reputation monitoring system that aggregates feedback for a user across the internet.
- Users can only retrieve components from verified providers.
- Provide a system for applications to identify all components in use
- Provide a system to monitor the security of all components in public databases, mailing lists, and security mailing lists, and keep them up to date.
- Provide a system to monitor underground channels that sell exploit code and take down these channels as they are identified. This can be difficult to achieve since the code of starting a new underground channel is relatively low.
- Require security testing of all components in use
- Design applications using appropriate security methodologies (Uzunov et al. 2012).

3.7 Forensics

Where can we find evidence of this attack?

- Providers can keep logs of the users that publish/retrieve components.
- We can audit a compromised application.

3.8 Related Patterns

Misuse patterns for cloud computing: Malicious virtual machine creation [K Hashizume et al. 2011]

4. DIRECT ACCESS TO OBJECTS USING UNCONTROLLED REFERENCES

4.1 Intent

Most applications utilize internal objects such as database and files as part of their functionality. In some cases, these internal objects are referenced directly through URL parameters and are referenced insecurely without proper authorization checks. A malicious user may take advantage of these insecure direct object references to access and manipulate unauthorized data (OWA10). This might lead to serious problems in business processes.

4.2 Context

Due to the expansion of the Internet, web applications are being developed using a variety of diverse technologies such as Java, .NET, PHP etc. Time, financial constraints, and lack of web developers' security knowledge vs. the rapid growth of technology along with increasing complexity in web attacks and hackers' tactics, have exposed web applications in many risks [Fonseca et al. 2009].

4.3 Problem

To perform some types of misuse it is necessary to identify an application that provides an insecure direct object reference for an attacker to misuse.
The attack can be performed by taking advantage of the following vulnerabilities:

- Often applications need to reference internal data e.g. database or file objects. It is convenient for developers to reference these objects directly using an internal identifier or name.
- In some cases, the applications do not check for proper authorization before providing access to internal data

4.4 Solution

Web applications are being deployed to the Internet and are accessible to users. This application might provide an insecure reference to internal data. An attacker can misuse this exposure to compromise the application and/or other applications.

4.4.1 Structure

Figure 5 shows a class diagram for the Insecure Direct Object Reference threat pattern.

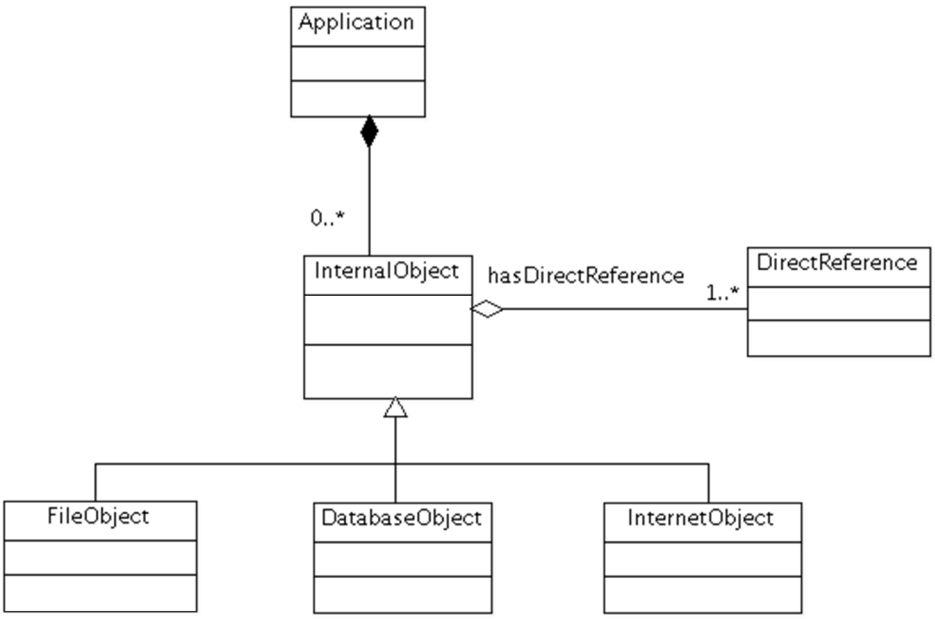

Figure 5. Class Diagram of the Insecure Direct Object Reference Threat Pattern

4.4.2 Dynamics

UC1: Compromise an Application with IDOR (Fig. 6)
Summary: The Attacker compromises an application with IDOR.

Actor: Attacker

Precondition: An application available to the attacker must provide insecure direct references to internal objects.

Description:

- The attacker requests unauthorized internal data. Since the application does not properly control access to internal data the attacker's request is successful.
- The attacker tries to modify the unauthorized data. Since the application does not properly control access to internal data the attacker's request is successful.

Postcondition:
The attacker accesses and/or modifies unauthorized internal data.

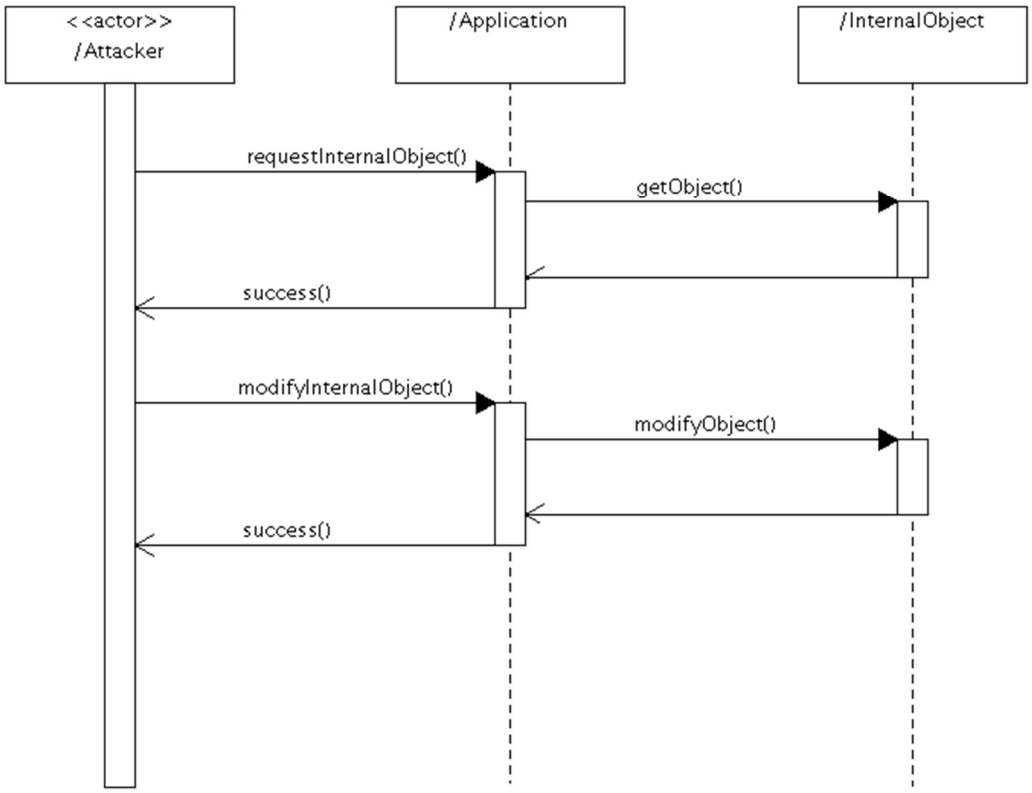

Figure 6. Sequence Diagram for the use case Compromise an Application with IDOR

4.5 Consequences

Some of the benefits of the misuse pattern are the following:

- An attacker can access and/or modify unauthorized data.
- Some data might contain credentials to other applications. This can be used to compromise other applications

Possible sources of failure include:

- The application may not have permissions to access files on the file system.
- Another module outside the application might enforce proper authorization controls.

4.6 Countermeasures

Compromising application with IDOR can be stopped by the following countermeasures:

- Use secure programming practices for development.
- Require security testing of all applications before being made available on the Internet.
- Design applications using appropriate security methodologies [Uzu12].

4.7 Forensics

Where can we find evidence of this attack?

- The application web logs.
- We can audit a compromised application.

5. DISCUSSION AND CONCLUSIONS

Designers need to understand first 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 them. 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 threats 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. 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 misuse patterns would give them a way to prove a degree of resistance to misuses without having to show their security details.

We can describe application security threats as patterns, which describe in a systematic way that application misuses are performed. 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 Takashi Kobayashi for his useful comments. The National Institute of Informatics of Japan paid the expenses for the second author to attend AsianPLoP.

REFERENCES

R. Böhme, Vulnerability markets. What is the economic value of a zero-day exploit? Proceedings of 22nd Chaos Communication Congress, (Berlin, Germany, dec. 27-30, 2005).

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.

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

J. Fonseca, M. Vieira, and H. Madeira, "Vulnerability & attack injection for web applications," in Dependable Systems Networks, 2009. DSN '09. IEEE/IFIP International Conference on, 2009, pp. 93 –102.

K Hashizume, E. B. Fernandez, and N. Yoshioka, "Misuse patterns for cloud computing: Malicious virtual machine creation"", Procs. of the Twenty-Third International Conference on Software Engineering and Knowledge Engineering (SEKE 2011), Miami Beach, USA, July 7-9, 2011

Ghafari, M.; Shoja, H.; Amirani, M.Y. "Detection and Prevention of Data Manipulation from Client Side in Web Applications", *Trust, Security and Privacy in Computing and Communications (TrustCom), 2012 IEEE 11th International Conference on,* On page(s): 1132 – 1136

Chris Grier , Lucas Ballard , Juan Caballero , Neha Chachra , Christian J. Dietrich , Kirill Levchenko , Panayiotis Mavrommatis , Damon McCoy , Antonio Nappa , Andreas Pitsillidis , Niels Provos , M. Zubair Rafique , Moheeb Abu Rajab , Christian Rossow , Kurt Thomas , Vern Paxson , Stefan Savage , Geoffrey M. Voelker, Manufacturing compromise: the emergence of exploit-as-a-service, Proceedings of the 2012 ACM Conference on Computer and Communications Security, October 16-18, 2012, Raleigh, North Carolina, USA

OWASP, "The Ten Most Critical Web Application Security Risks." 2010. https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

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

A. V. Uzunov, E.B.Fernandez, and K. Falkner, "Engineering Security into Distributed Systems: A Survey of Methodologies", Journal of Universal Computer Science, Vol. 18, No. 20, 2012, pp. 2920-3006. http://www.jucs.org/jucs_18_20/engineering_security_into_distributed

M.I.Yague, ; A. Mana, J. Lopez, J.M. Troya, "Applying the semantic Web layers to access control," *Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on* , vol., no., pp.622,626, 1-5 Sept. 2003