|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | |  | | KITRI 모의해킹 28기 | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  |  | | |  | |
|  | 3.Sensitive Data Exposure | | | | | | |  |
|  | | (민감한 데이터 노출) | | | |  | | |
|  | |  | | | |  | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |

|  |
| --- |
| 1. 개요 |
| https://owasp.org/www-project-top-ten/2017/A3\_2017-Sensitive\_Data\_Exposure  개인 식별 정보나 신용 정보가 대표적인 민감 데이터라고 할 수 있다.  웹 어플리케이션마다 중요한 민감 데이터를 취급하고 있을 것이며 이를 올바르게 보호하지 않는다면(프로그램이 보안과 관련된 민감한 데이터를 평문으로 송·수신할 경우, 스니핑을 통해 민감한 데이터가 노출될 수 있다.) 공격자에게 노출될 수 있다.  민감한 데이터 노출은 저장되어 있는 데이터일 수도 있고 전송중인 데이터일 수도 있다. (데이터를 평문 전송하는 것이 그 예이다.)  현재에도 많은 웹 어플리케이션의 암호화되어 있지 않는 평문 데이터들이 공격자에게 노출되어 있을 것이다. |
| ***\*\*\* 비박스를 설치한 상태여야 함 \*\*\**** |
| References |
| OWASP  \* OWASP Proactive Controls: Protect Data Everywhere  \* OWASP Application Security Verification Standard (V7, 9, 10)  \* OWASP Cheat Sheet: Transport Layer Protection  \* OWASP Cheat Sheet: User Privacy Protection  \* OWASP Cheat Sheet: Password and Cryptographic Storage  \* OWASP Cheat Sheet: HSTS  \* OWASP Testing Guide: Testing for weak cryptography  External  \* CWE-202: Exposure of sens. information through data queries  \* CWE-310: Cryptographic Issues  \* CWE-311: Missing Encryption  \* CWE-312: Cleartext Storage of Sensitive Information  \* CWE-319: Cleartext Transmission of Sensitive Information  \* CWE-326: Weak Encryption  \* CWE-327: Broken/Risky Crypto  \* CWE-359: Exposure of Private Information (Privacy Violation) |

|  |
| --- |
| 1. Base64 Encoding (Secret) |
| **base64** : 8bits 2진 데이터를 문자 코드에 영향을 받지 않는 공통 ASCII 영역의 문자들로만 이루어진 일련의 문자열로 바꾸는 인코딩 방식이다. 인코딩이기 때문에 당연히 디코딩이 가능하며, 중요 데이터를 base64로 인코딩하여 보호하는 것은 보안에 취약하다.    **해시(Hash) 함수** : 임의의 길이 N을 갖는 데이터를 고정된 길이의 M으로 매핑하는 함수. 이론상으로 역으로 되돌리는 것이 불가능하여 보안상 안전하지만, 복호화가 가능한 데이터도 존재한다. (문자열 A가 해시하면 해시 값 B라는 것을 알면, 해시 값 B가 문자열 A라는 것을 알 수 있기 때문.)  그리고 보통 해시 함수는 암호화/복호화한다고 말하지 않는다. 왜냐하면 해시 함수 설계 자체가 역으로 되돌리는 것(복호화)이 불가능하게 설계되었기 때문이다.  \* 해시 함수 종류 : MD5, SHA-1, SHA-2(SHA-256) 등 |
| 실습 |
| **[ Sensitive Data Exposure - Base64 Encoding (Secret) <난이도 : 하> ]**    크롬에서 editcookie로 열면    문제가 base64 인코딩 문제이므로, base64 디코더 사이트에서 해당 문자열을 디코딩해봤다.  이때 인코딩된 값에서 "%2F"는 "/"(슬래시)이다. 이는 HTTP 패킷으로 전송되면서 URL 인코딩이 되었기 때문이다.    위와 같이 base64 방식으로 인코딩된 중요 정보는 쉽게 디코딩되어 그 값을 알 수 있다.    **[ Sensitive Data Exposure - Base64 Encoding (Secret) <난이도 : 중/상> ]**    난이도 중/상에서 쿠키를 확인해보면, 난이도 하와는 그 값이 다르다.        이 값을 base64 디코딩을 해보면 오류가 발생한다. 이는 다른 방식으로 암호화가 되었다는 건데,  해시 함수라고 가정하고 어떤 해시 함수로 해시되었는지 알아보겠다.    칼리 리눅스에 hash-identifier이라는 도구가 있다. 이 도구는 해시 값을 입력하면 어떤 종류의 해시인지 추측해주는 도구이다.  이 도구를 사용해보자.  **Kali > hash-identifier**    인 것을 알아낼 수 있다.  또한 적은 가능성으로 가능한 해시 함수의 목록도 확인할 수 있다.    SHA-1은 MD5 보다 조금 더 안전하지만, 현재 MD5와 SHA-1 알고리즘의 결함이 발견되었지만 사용을 못할 정도는 아니다.    SHA-1도 일부 복호화(역해시)가 가능하다.    구글링하면 해시 함수를 복호화할 수 있는 사이트를 쉽게 찾을 수 있다.    \* SHA-1 해시/역해시 사이트 : <https://md5hashing.net/hash/sha1>  \* SHA-1 해시/역해시 사이트 : <https://hashkiller.co.uk/Cracker/SHA1> |

|  |
| --- |
| 2. BEAST/CRIME/BREACH Attacks |
| **BEAST(Browser Exploit Against SSL/TLS)**:  - CBC(암호 블록 체인)의 약점을 이용해 암호화된 세션에서 암호화되지 않은 일반 텍스트를 추출하는 공격 방식  - 관련 CVE 번호 : [CVE-2011-3389](https://www.cvedetails.com/cve/CVE-2011-3389/)    **CRIME(Compression Ration Into-Leak Mass Exploitation)**  - HTTP의 압축의 취약점을 이용해 암호화된 HTTP 요청 패킷을 복구하여 쿠키를 훔쳐 세션을 가로채는 공격 기법  - 관련 CVE 번호 : [CVE-2012-4929](https://www.cvedetails.com/cve/CVE-2012-4929/)      **BREACH(Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext)**  - CRIME exploit 기법을 기반으로 하는 공격 기법으로, CRIME 공격과 비슷한 공격 기법이다.  - 이 공격 또한 HTTP의 압축의 취약점을 이용하여 HTTP 응답 패킷을 악용하는 공격이다.  - 이 응답 패킷은 일반적인 HTTP 압축 매커니즘을 사용하여 압축되므로, SSL/TLS 버전에 의존하지 않고 모든 SSL/TLS 버전과 모든 암호 알고리즘에서 작동되어 더 강력한 공격 기법이다.    \* 참고 1 : <http://www.ciokorea.com/news/40172>  \* 참고 2 : <https://blogs.cisco.com/security/breach-crime-and-blackhat> |
| 실습 |
| **[ Sensitive Data Exposure - BEAST/CRIME/BREACH Attacks <공격> ]**      Lighttpd 웹 서버는 BEAST, CRIME, BREACH 공격에 취약하고,  TLS 버전을 1.1 이상으로 구성하고, SSL과 HTTP 압축 기능을 해제하라고 한다.    그리고 O-Saft 도구를 이용해 9443번 포트로 SSL 연결을 테스트해보라고 하고 있다.    실제로 BEAST, CRIME/BREACH 공격을 하기에는 무리가 있을 거 같고(공격 방법을 연구하는데 오래 걸릴 거 같다...), 여러 SSL 스캔 도구를 사용을 해보겠다.       |  | | --- | | **O-Saft 도구**를 간략히 소개하자면  OWASP 프로젝트 중 하나로, SSL 포렌식 / 감사 도구이다.  SSL 인증서에 대한 정보를 보여주고, 다양한 SSL 구성과 암호 목록으로 SSL 연결을 테스트해볼 수 있다.  \* 참고 사이트 1 (O-Saft 소개) : <https://www.owasp.org/index.php/O-Saft>  \* 참고 사이트 2 (O-Saft 사용자 문서) : <https://www.owasp.org/index.php/O-Saft/Documentation> |   칼리 리눅스에 다운로드 받는다.  **Kali > wget**[**https://github.com/OWASP/O-Saft/archive/19.01.19.tar.gz**](https://github.com/OWASP/O-Saft/archive/19.01.19.tar.gz)  다운로드 후 압축을 풀어준다.  **Kali > tar -xvf 19.01.19.tar.gz**    이제 압축이 풀린 파일의 디렉터리로 가서 O-Saft 도구를 사용해보자.    먼저 도움말은 다음 명령어를 사용해 확인한다.  **Kali > ./o-saft.pl --help**    사용 형식과 주요 명령은 다음과 같다.   |  | | --- | | \* 형식 :**./o-saft.pl [+option] [target\_domain | target\_IP]:[Port]**    **[o-saft 정보]**  **+list**​: 이 도구에서 지원하는 모든 암호 목록을 출력  **+version** : 버전 정보 출력    **[SSL 연결에 대한 세부 사항]**  **+info** : SSL 연결에서 중요한 정보를 출력  **+http** : HTTP 검사  **+check** : 보안 문제와 관련한 SSL 연결을 점검    **[대상 서버에서 제공하는 암호 테스트]**  **+cipherall**: 해당 SSL에서 사용 가능한 암호를 확인 |   먼저 SSL 연결의 중요 정보를 출력해주는 +info 명령부터 사용해보자.  **Kali > ./o-saft.pl +info [BEE-BOX\_IP]:9443**    위와 같이 비박스 서버의 SSL 연결 정보를 확인할 수 있다.    다음으로 +check 명령을 사용한다.  **Kali > ./o-saft.pl +check [BEE-BOX\_IP]:9443**    암호 체크 부분에서 첫 번째 항목은 암호의 이름이고, 두 번째 항목은 해당 서버에서 해당 암호를 지원하는지, 세 번째 항목은 보안의 강도를 의미한다.    다음으로 +cipherall 명령을 사용한다.  **Kali > ./o-saft.pl +cipherall [BEE-BOX\_IP]:9443**      위와 같이 비박스 SSL 서버에서 사용 가능한 암호 목록을 확인할 수 있다.    다음으로 칼리 리눅스에서 제공하는 sslscan 도구와 sslyze 도구를 사용해보겠다.    sslscan 도구의 사용법은 간단하다.  **Kali > sslscan [BEE-BOX\_IP]:9443**    위와 같이 명령어 수행 결과를 확인할 수 있다.    결과를 보면 TLS Compression 항목에서 압축 기능이 활성화되어 있다는 것을 확인할 수 있고,  CRIME 공격이 가능함을 파악할 수 있다.    다음으로 sslyze 도구를 사용해보겠다.  **Kali > sslyze --regular [BEE-BOX\_IP]:9443**       3개의 취약점을 발견하였다. |
| **[ Sensitive Data Exposure - BEAST/CRIME/BREACH Attacks <보안 대책> ]** |
| BEAST, CRIME/BREACH와 같은 SSL 취약점을 이용한 공격을 방어하기 위해서는 최신 버전의 SSL을 사용하고, 강력한 암호 조합을 사용한다. 그리고 HTTP 압축 기능을 비활성하는 등의 방법이 있다.    공격 실습 참고 : <http://breachattack.com/> / <https://github.com/nealharris/BREACH>  - 여기서 BREACH 테스트 도구를 제공하고 있다. |

|  |
| --- |
| 3. Clear Text HTTP (Credentials) |
| Clear Text 란, 암호화가 되지 않고 전송되거나 저장된 데이터를 뜻한다.  - HTTP는 평문으로 데이터를 전송하기 때문에 스니핑에 의해서 정보가 유출될 수 있다.  - 그렇기 때문에 SSL(HTTPs)를 사용하여 보안 통신 연결 이후, 중요한 정보는 암호화/인증을 실시하여 전송해야 한다. |
| 실습 |
| **[ Sensitive Data Exposure - Clear Text HTTP (Credentials) <난이도 : 하> ]**  칼리 IP(희생자) : 192.168.219.104  비박스 IP : 192.168.219.103      먼저 칼리 리눅스에서 WireShark를 실행한다.  **Kali > wireshark &**    패킷을 스니핑할 네트워크 인터페이스를 선택해준 후, 아래처럼 필터링을 걸어준다.  **wireshark filter > ip.addr==[희생자\_IP] && ip.addr==[비박스\_IP]**    이제 비박스 문제 페이지에서 로그인을 하고 패킷을 보겠다.  위와 같이 TCP와 HTTP 프로토콜의 패킷을 볼 수 있다.    이제 해당 패킷의 세부 내용을 보자.        아무 패킷에서 [Follow] - [HTTP Stream]으로 들어간다.    위와 같이 HTTP 요청/응답 패킷의 내용을 확인할 수 있는데, 평문으로 노출된 ID와 PW를 확인할 수 있다.      **[ Sensitive Data Exposure - Clear Text HTTP (Credentials) <난이도 : 중/상> ]**      난이도 하와 다르게 아래 텍스트가 하나 출력된다.  인증서는 반드시 완전하게 구성되어야 한다고 한다.    일단 난이도 하와 똑같이 공격을 진행해보자.    WireShark를 패킷 캡처 모드로 해두고, 비박스 문제 페이지에서 로그인을 시도한다.        로그인을 하면 위와 같은 경고창이 뜬다.  이는 비박스 서버가 자체 SSL 인증서를 사용하기 때문이다.    [고급] - [안전하지 않음]을 클릭해서 넘겨준다.  이제 WireShark에서 패킷을 확인해보면      위와 같이 TCP와 TLS 패킷을 확인할 수 있는데 이 TLS 패킷은 암호화된 패킷으로, 내용이 암호화되어 확인할 수 없다.    로그인 후에 URL도 위와 같이 "https://"로 바뀌었음을 확인할 수 있다.    이제 서버 내 코드를 확인해보겠다.  #vi /var/www/bWAPP/insuff\_transp\_layer\_protect\_1.php    난이도 하(security\_level=0)에서는 $url 변수에 들어가는 값을 현재 파일 이름으로 저장하고 있고,  난이도 중/상(security\_level=1/2)에서는 $url 변수에 들어가는 값을 "https://"를 포함한 현재 URL로 저장하고 있다.  \* **$\_SERVER["SCRIPT\_NAME"]** : 현재 파일 이름  \* **$\_SERVER["HTTP\_HOST"]** " : 서버 URL    이 $url 변수는128라인에서 사용된다.          form 태그의 action 속성으로 $url 변수가 들어가는 것을 확인할 수 있다. |

|  |
| --- |
| 4. Host Header Attack (Reset Poisoning) |
| **Host Header Attack / Reset Poisoning** : 자세한 설명은 아래 링크 참조  <https://www.acunetix.com/blog/articles/password-reset-poisoning/>  <https://www.acunetix.com/blog/articles/automated-detection-of-host-header-attacks/> |
| 실습 |
| **[ Sensitive Data Exposure - Host Header Attack (Reset Poisoning) <난이도 : 하> ]**    Email을 입력하면 입력한 이메일 주소로 secret을 초기화할 수 있는 reset code가 포함된 메일을 보내주는 페이지이다.    테스트 계정을 만든 후, 계정 생성할 때 등록한 이메일 주소를 입력하고 [Reset]을 클릭하면 아래처럼 리셋 코드를 이메일로 보냈다는 메시지가 출력된다.    하지만 실제로 이메일 보내는 기능이 이 페이지에 없는 것 같다.   실습을 진행하기 위해서 비박스 서버에서 해당 페이지의 코드를 조금 변경했다.  # vi /var/www/bWAPP/hostheader\_2.php        위 파일에서 $message 변수는 사용자가 입력한 이메일이 DB에 존재하는지 확인한 결과를 문자열로 저장한 변수이다.  아까 위에서 봤던 초록색 글씨로 출력된 텍스트가 이 $message 변수에 담긴 내용이다.    $content 변수는 reset을 요청한 사용자 메일로 보낼 메일의 내용이 담긴 문자열이 저장되어 있다.    이를 문제 페이지 상에 출력하도록 코드를 변경한 것이다.    이제 다시 해당 페이지에서 이메일을 입력하고 [Reset]을 클릭해본다.    아래에 출력되는 텍스트는 실제 사용자가 받는 메일의 내용이라고 가정한다.    그리고 이제, Host Header Attack을 진행해보겠다.   |  | | --- | | **비박스 IP [희생자가 이용하는 웹 서버]: 192.168.219.103**  **칼리 리눅스 IP [공격자가 사용하는 웹 서버] : 192.168.219.104**    \* 네트워크는 bridge로 연결    **공격 시나리오** : Host Header Attack에 취약한 웹 서버를 대상으로 희생자 이메일 계정의 secret을 초기화(reset)하는 요청을 위조하여, 정상적인 secret 변경 페이지가 아닌  공격자가 임의로 만든 페이지의 링크를 희생자의 이메일 계정으로 보내, 희생자가 이 링크를 클릭하도록 유도한다. |   먼저 Host Header Attack의 취약점이 있는지 확인하기 위해 문제 페이지에서 [Reset] 버튼을 클릭했을 때의 요청 패킷을  캡처한다, 패킷 캡처에는 Burp Suite를 사용한다.    HTTP 요청 패킷에서 **Host 헤더는 접속을 요청하는 사이트**이다.  위 패킷을 보면 비박스의 IP로 지정되어 있는 것을 볼 수 있다.    이제 이 Host 헤더를 변조해보겠다.    Host 헤더의 내용을 "kitri\_attacker.co.kr"로 변조하고 패킷을 통과시킨다.  그다음에 메일의 내용을 확인해봤다.    위처럼 메일 내용에서 secret을 변경할 수 있는 링크가 바뀐 것을 확인할 수 있다.    이를 이용해 특정 계정의 secret을 바꿀 수 있겠다.    먼저, 칼리 리눅스를 통해 사용자의 secret 변경을 위한 reset\_code를 탈취할 수 있는 페이지를 만든다.  (칼리 리눅스에 아파치와 PHP 설치가 선행되어야 한다.)  아파치 서버를 구동시킨 후, 웹 루트 디렉터리에 bWAPP 디렉터리를 생성해준다.  (이메일로 보내는 링크의 디렉터리가 bWAPP이기 때문에)  **Kali > service apache2 start**  **Kali > mkdir bWAPP**    그리고 탈취한 사용자의 reset\_code를 저장하기 위해 파일 입출력을 할 것이므로 bWAPP 디렉터리의 소유자를 바꾸어준다.  (안 바꾸면 권한 오류로 파일을 생성/쓰기가 불가능하다.)  **Kali > chown www-data:www-data bWAPP/**  그리고 링크와 똑같이 secret\_change.php 파일을 만들어준다.  **Kali > vi secret\_change.php**  <?php  $email = $\_GET["email"];  $token = $\_GET["reset\_code"];  $file = fopen("token.txt", "a");  fwrite($file, $email." ".$token);  fclose($file);  echo "hacked by kitri";  ?>  위와 같이 간단한 PHP 코드를 작성해준다.    간단하게 코드 해석을 하면, GET으로 받은 email, reset\_code 파라미터 변수를 각각 변수에 저장한 뒤에,  token.txt 파일에 해당 변수를 저장하는 코드이다.    사용자의 reset code를 탈취하기 위한 페이지가 만들어졌으니, 이제 사용자 이메일 계정에 위조된 링크가 포함된 메일을 보내면 된다.    비박스 문제 페이지에서 사용자의 이메일을 입력하고 [Reset] 버튼을 클릭할 때의 요청 패킷을 캡처한다.    위와 같이 공격자의 서버의 링크를 사용자에게 보내게 된다.    디렉터리, 파일 이름까지 똑같이 만들었기 때문에 만약 희생자가 이 링크를 클릭하게 된다면 희생자의 secret을 초기화하고 변경할 수 있는 reset\_code를 얻게 되는 것이다.    실제로 저 페이지를 접속하면 아래와 같이 임의로 만든 페이지에 접속할 수 있다.  (실습이기 때문에 간단하게 만들었지만, 실제 공격에 사용된다면 실제 secret\_change.php 페이지와 똑같이 만들고 동작하는 것처럼 만들면, 사용자는 공격을 당한 사실도 모를 것이다.)    그리고 공격자의 서버<Kali>에서 파일을 확인해보면  **Kali > cat token.txt**      위와 같이 희생자가 페이지를 접속하는 것으로 희생자의 secret을 초기화하고 변경할 수 있는 reset\_code를 얻을 수 있는 것을 확인했다.    만약에 실제 웹사이트에서 비박스 문제 페이지처럼 이메일로 패스워드를 초기화하고 변경할 수 있는 링크를 보내는 기능이 있다면, 공격자는 자신의 계정으로 Host Header Attack의 취약점이 있는지 확인해볼 수 있을 것이고, 취약점이 존재한다면 위장 페이지를 만들어서 다른 사용자의 패스워드 초기화하고 변경할 수 있는 링크를 얻을 수 있을 것이고, 이는 계정 탈취로 이어질 것이다.      **[ Sensitive Data Exposure - Host Header Attack (Reset Poisoning) <난이도 : 중/상> ]**  난이도 중/상에서도 Host Header Attack 공격을 수행해봤다.    위처럼 "attacker.co.kr"로 Host 헤더를 변조하고 패킷을 통과시킨다.      난이도 하와 다르게, 링크의 도메인이 "itsecgames.com"인 것을 확인할 수 있다.    비박스 서버 내 코드를 확인해보자.  # vi /var/www/bWAPP/hostheader\_2.php    난이도 하(security\_level=0)에서는 $server 변수의 값을 $\_SERVER["HTTP\_HOST"] 변수에서 받아오고 있고,  **난이도 중/상(security\_level=1/2)에서는 $server 변수의 값을 고정시켜 놓음으로써 변조를 방지하고 있다.**  **\* $\_SERVER["HTTP\_HOST"]**: PHP의 환경변수로, 접속을 요청한 사이트를 저장하는 HTTP 패킷의 Host 헤더의 값이다. |

|  |
| --- |
| 5. HTML5 Web Storage (Secret) |
| **HTML5 Web Storage** : 말 그대로 웹 저장소. 웹 서버에 저장하는 것이 아닌, 브라우저 로컬에 저장하는 저장소를 말한다.  이 웹 저장소는 Local Storage와 Session Storage 두 가지 종류가 있다.  쿠키(Cookie)와 비슷한 개념이지만 약간의 차이가 있다.    **쿠키와 다른 점**은, 데이터의 유효기간과 용량 차이이다. 쿠키의 경우에 유효기간이 존재하고 저장할 수 있는 데이터의 양도 제한이 있지만, 이 웹 저장소는 유효기간의 제한이 없고 쿠키보다 저장할 수 있는 데이터의 양도 크다.    **Local Storage와 Session Storage의 차이**는 데이터 유효기간의 차이이다. Local Storage의 경우에는 기본적으로 의도적으로 삭제하지 않으면 계속 브라우저에 남아있고, Session Storage는 세션이 유지되는 동안에만 유효하다. |
| 실습 |
| **[ Sensitive Data Exposure - HTML5 Web Storage (Secret) <난이도 : 하> ]**        나의 로그인 이름과 secret이 HTML5 웹 저장소에 저장되어 있다고 한다.  그리고 XSS 기법을 사용해 이 정보를 알아내라고 하고 있다.    Local Storage/Session Storage는 사용자 브라우저에 저장되기 때문에, 사용자가 확인할 수 있다.    이 웹 저장소의 값은 크롬의 개발자 도구(F12)에서 확인할 수 있다.  개발자 도구에서 [Storage] 탭으로 들어간다.    Local Storage에서 해당 페이지 주소를 클릭하면 값을 확인할 수 있다.      위와 같이 Local Storage에 secret과 login 키에 저장되어 있는 값을 확인할 수 있다.    이제 XSS 기법을 이용해서 Local Storage에 저장된 값을 알아내기 위해 비박스의 [XSS - Reflected (GET)](https://blog.naver.com/is_king/221596700164) 페이지에서 스크립트 코드를 삽입해보겠다.      위 페이지에서 스크립트를 삽입하여 Local Storage 값을 가져올 수 있다.    First name 폼에 다음과 같은 코드를 삽입하고, Last name에는 아무 값이나 입력해준다.   |  | | --- | | <script>      var str = "";        for(var key in localStorage) {          str += "<br>" + key + " = " + localStorage.getItem(key);      }        document.write(str);  </script> |   **\*** **localStorage.getItem(key)** : 해당 key의 값을 가져옴    **\* localStorage.setItem(key, value)**  **\* localStorage.key = value** : 해당 'key'에 'value' 값을 저장          위처럼 secret 값과 login 값을 알아낼 수 있다.    secret과 같은 중요한 정보를 쿠키나 웹 저장소에 저장하게 되면 위와 같이 스크립트 삽입을 통해서 노출이 될 수 있다.      **[ Sensitive Data Exposure - HTML5 Web Storage (Secret) <난이도 : 중/상> ]**  난이도 중/상에서도 개발자 도구를 통해 웹 저장소의 값을 확인해봤다.    난이도 하와 다르게 secret 값이 암호화가 되어 있다.    이 값은 SHA-1으로 해시되어 있는데, SHA-1은 복호화가 가능하다.  **SHA-1 복호화 사이트 :** <https://hashkiller.co.uk/Cracker/SHA1>          위와 같이 복호화가 되어 원래의 값을 확인할 수 있다.    이제 서버 내 코드를 확인해보자.      (/var/www/bWAPP/insecure\_crypt\_storage\_1.php)    위는 Local Storage에 데이터를 저장하는 부분의 자바스크립트 코드이다.    난이도 하(security\_level=0)일 때는 평문 상태의 secret 값을 Local Storage에 저장하고 있다.  난이도 중/상(security\_level=1/2)일 때는 SHA-1으로 해시하여 저장하고 있는 것을 볼 수 있다.    위와 같이 해시를 해도 복호화가 가능할 수 있기 때문에, XSS 같은 공격 기법으로 스크립트 삽입을 통해 노출이 되지 않도록 웹 취약점을 보완해야 한다. |

|  |
| --- |
| 6. POODLE Vulnerability |
| - POODLE : Padding Oracle On Downloaded Legacy Encryption   - SSLv3 프로토콜에서 암호화되어 통신 중인 암호문 중 일부가 평문으로 복호화가 가능한 취약점이다.   - 블락암호화 기법인 CBC(Cipher Block Chaning) 모드를 사용할 경우, 발생하는 패딩된 암호화 블락이     MAC(메세지 인증 코드)에 의해서 보호되지 않기 때문에 발생한다.   - CVE-2014-3566 취약점 : <https://cve.mitre.org/cgi-bin/cvename.cgi?name=cve-2014-3566> |
| 실습 |
| **[ Sensitive Data Exposure - POODLE Vulnerability <난이도 : 하> ]**  **'Exploiting The SSL 3.0 Fallback' 클릭 -> SSLv3 취약점 PDF가 열림**          **'nmap'을 이용하여 bWAPP POODLE 취약점 스캔 실시**    현재 취약한 상태라고 나오게된다. |
| **<보안 대책>** |
| OpenSSL에서 1.0.1i 및 기타 제품에 사용되는 SSL 프로토콜 3.0을 사용하지 않는다. |
| 7. Text Files(Accounts) |
| - 데이터를 평문으로 전송하면 스니핑에 의해서 유출되는 문제점이 있다.   - 그렇기 때문에 MD5, SHA-1, SHA-2를 사용하여 해시값을 통하여 안전하게 전송해야 한다.   - 단, MD5와 SHA-1은 현재 레인보우 테이블에 의해서 크랙이 가능하기 때문에 SHA-2를 사용해야 한다.   - 레인보우 테이블 참고 사이트 : <http://project-rainbowcrack.com/table.htm> |
| 실습 |
| **[ Sensitive Data Exposure - Text Files(Accounts) <난이도 : 하> ]**          **'Download' 클릭**          **bug 계정 패스워드 확인**      **[ Sensitive Data Exposure - Text Files(Accounts) <난이도 : 중/상> ]**  **각각의 난이도에서 아이디 비번을 삽입한다.**          **'Download' 클릭**      **bug 계정 패스워드 확인**    **각각 난이도에 따라 결과 값이 다르다, 소스를 열어보자**    난이도 중에서는 sha1을 통해 암호화가 되었고,  난이도 상에서는 sha2 (sha256)을 통하여 salt값을 추가해 암호화가 된 것을 확인할 수 있다.    **'hash-identifier'을 이용하여 해시 타입 확인** |
| <대응방법> |
| MD5와 SHA-1은 현재 레인보우 테이블에 의해서 크랙이 가능하기 때문에 SHA-2를 사용해야 한다.  따라서, 중요 정보 전송에는 sha 256이상을 사용하여 암호화한 뒤 전송하도록 한다. |

|  |
| --- |
| 결론 및 대응방법 |
| **1. 인증 정보와 같은 민감한 정보 전송시 안전하게 암호화해서 전송해야 한다.**   : 분석단계에서 정의된 중요정보를 네트워크를 통해 전송해야 하는 경우 안전한 암호모듈로 암호화 한 뒤 전송하거나 안전한 통신 채널을 사용하도록 설계해야 합니다. 안전한 암호화는 “암호연산” 요구 항목을 충족시키는 암호화 알고리즘이나 암호키를 사용해야 합니다.    **2. 쿠키에 포함되는 중요 정보는 암호화해서 전송해야 합니다.**  : 쿠키에는 중요정보가 포함되지 않도록 설계해야 하지만 부득이 쿠키에 중요정보가 포함되어야 하는 경우에는 반드시 세션 쿠키로 설정되어야 하며, 전달되는 중요 정보는 반드시 암호화해서 전송해야 한다. |