ISBN : 000-00-000-0000-0

웹 어플리케이션 취약점 분석
('17년 1분기)

2017. 03

KiSTi www.kisti.re.kr
한국과학기술정보연구원
Korea Institute of Science and Technology Information

웹 어플리케이션 취약점 분석
('17년 1분기)

2017. 03

부 서 : 첨단연구망센터 첨단연구망정보보호실
제출자 : 이형주 (lhj275@kisti.re.kr)

KiSTi www.kisti.re.kr 한국과학기술정보연구원 Korea Institute of Science and Technology Information

[목 차]

제1장 서론

최근 웹을 이용한 침해사고를 미연에 방지하기 위한 하나의 방법으로써 웹 취약점 분석에 관한 연구가 활발히 진행되고 있다. 네트워크 환경이 실 생활에 필수 요소로 자리 잡은 지금 웹은 모든 응용 계층 중에 가장 많이 사용하는 프로토콜이 되었다. 이러한 환경의 변화로 많은 양의 웹 응용 프로그램들이 등장하게 되었고, 이들의 취약점을 이용한 공격사례들이 증가하게 되었다.

웹 서비스는 개방된 환경에서 보안장비를 거치치 않고 사용자와 서버 간 통신이 연결되는 구조적으로 취약한 특성을 가지고 있어 이로 인해 악의적인 공격자에 의한 공격타겟이 되기 쉽다. 이러한 공격을 보호하기 위한 대책으로 보안장비 도입을 통한 실시간 모니터링, 보안정책 관리 등의 보안조치를 수행하고 있다. 하지만 이러한 보안시스템들은 웹 어플리케이션 취약점의 근원적인 문제 해소를 보장하지 못한다. 따라서 날로 지능화되는 공격기법에 대응하기 위해서는 웹 어플리케이션에 대한 지속적인 취약점 점검 및 개선조치가 반드시 필요한 실정이다.

이에, 과학기술사이버안전센터에서는 선제적인 웹 취약점 제거를 통해 보안사고를 미연에 방지할 수 있도록 자동화기반의 취약점 진단 시스템을 구축·보급하여 대상기관에서 운영중인 웹사이트의 균형적인 보안수준 향상을 도모하고 있다.

본 기술보고서에서는 17년도 1/4분기에 취약점 블랙박스 테스트를 통해 주로 탐지된 『**실명인증 취약점, 전송 시 개인정보 노출 취약점, 파일 다운로드/업로드 취약점**』을 중심으로 취약점에 대한 상세한 설명과 취약점 개선에 필요한 조치방안을 기술하고자 한다.

제2장 관련 연구

제1절 웹 어플리케이션 취약점 유형

본 절에서는 웹 어플리케이션에서 발생하는 취약점의 정의와 주요 탐지 유형에 대하여 살펴본다. 과학기술사이버안전센터에서 정의한 17개의 취약점 유형은 아래와 같다.

1 관리자 페이지 노출 취약점

일반적으로 추측이 가능한 관리자 페이지 경로(/admin, /manager 등)를 사용하거나, 프로그램 설계상의 오류, 인증 미흡으로 인해 관 리자 메뉴에 직접 접근이 가능하며 권한인증이 가능한 취약점

2 디렉터리 나열 취약점

서버내의 모든 디렉터리 혹은 중요한 정보가 포함된 디렉터리에 대해 인덱싱이 가능하게 설정되어 중요파일 정보가 노출될 수 있는 취약점

3 시스템 관리 취약점

응용 프로그램 설치 중에 생성되는 설치·임시 파일이 존재하거나 웹상에서 윈도우 로그인 창이 노출되는 등 시스템 상 설정 미비로 인해 발생하는 취약점

4 WEBDAV 취약점

IIS 일부 버전의 취약점으로 악의적인 HTTP 요청을 이용하여 FTP나 시스템에 직접 접근하지 않고 원격에서 파일을 수정 및 처리가 가능한 취약점

5 불필요한 Method 허용 취약점

웹 서비스 제공 시 불필요한 Method(PUT, DELETE, OPTIONS 등) 허용으로 외부 공격자에 의해 악성파일을 업로드 하거나 중요파일에 대한 조작이 가능해지는 취약점

6 취약한 파일 존재 취약점

웹 루트 하위에 내부 문서나 백업파일, 로그파일, 압축파일과 같은 파일이 존재할 경우 파일명을 유추하여 파일명을 알아내고, 직접 요청하여 해킹에 필요한 서비스 정보를 획득할 수 있는 취약점

7 계정 관리 취약점

회원가입 시에 안전한 패스워드 규칙이 적용되지 않아서 취약한 패스워드로 회원 가입이 가능할 경우 무차별 대입공격을 통해 패스워드가 누출될 수 있는 취약점

8 실명인증 취약점

본인 확인 과정상에서 취약한 프로그램을 악용하여 사용자정보를 변조하는 공격으로 관리자 위장을 통해 개인정보를 수집하거나 기타 공격에 악용할 수 있는 취약점

9 전송 시 개인정보 노출 취약점

프로그램이 보안과 관련된 민감한 데이터를 평문으로 통신 채널을 통해서 송·수신 할 경우, 통신채널 스니핑을 통해 인가되지 않은 사용자에게 민감한 데이터가 노출될 수 있는 취약점

10 파일 다운로드 취약점

외부 입력값에 대해 경로 조작에 사용될 수 있는 문자를 필터링하지 않는 취약점을 악용하여 예상 밖의 접근 제한 영역에 대한 경로 문자열 구성이 가능해져 시스템 정보누출, 서비스 장애 등을 유발 시킬 수 있는 취약점

11 파일 업로드 취약점

공격자가 웹 사이트에 있는 게시판이나 자료실의 파일 업로드 기능을 이용하여 공격자가 만든 특정 공격 프로그램을 업로드하여 웹 서버의 권한 획득이 가능한 취약점

12 소스코드 내 중요정보 노출 취약점

소스코드 주석문에 민감한 정보(개인 정보, 시스템 정보 등)이 포함되어 있는 경우, 외부 공격자에 의해 패스워드 등 보안 관련정보가 노출될 수 있는 취약점

13 공개용 웹 게시판 취약점

공개용 게시판을 사용할 경우 인터넷에 공개된 각종 취약점 정보로 인해 홈페이지 변조 및 해킹 경유지로 사용될 수 있는 취약점

14 크로스사이트스크립트(XSS) 취약점

공격자가 클라이언트 스크립트를 악용하여 웹사이트에 접속하려는 일반 사용자로 하여금 공격자가 의도한 명령이나 작업을 수행하는 공격으로, 세션탈취, 웹사이트 위변조, 악성 스크립트 삽입 및 실행, 접근경로 리다이렉트 등의 다양한 공격을 유발할 수 있는 취약점

15 구문삽입(SQL-Injection) 취약점

URL 요청 또는 웹 요청에 포함되는 웹 어플리케이션에서 입력 폼 및 URL입력란에 SQL 문을 삽입하는 형태의 공격으로 시스템 내부정보를 열람 또는 조작할 수 있는 취약점

16 권한인증 취약점

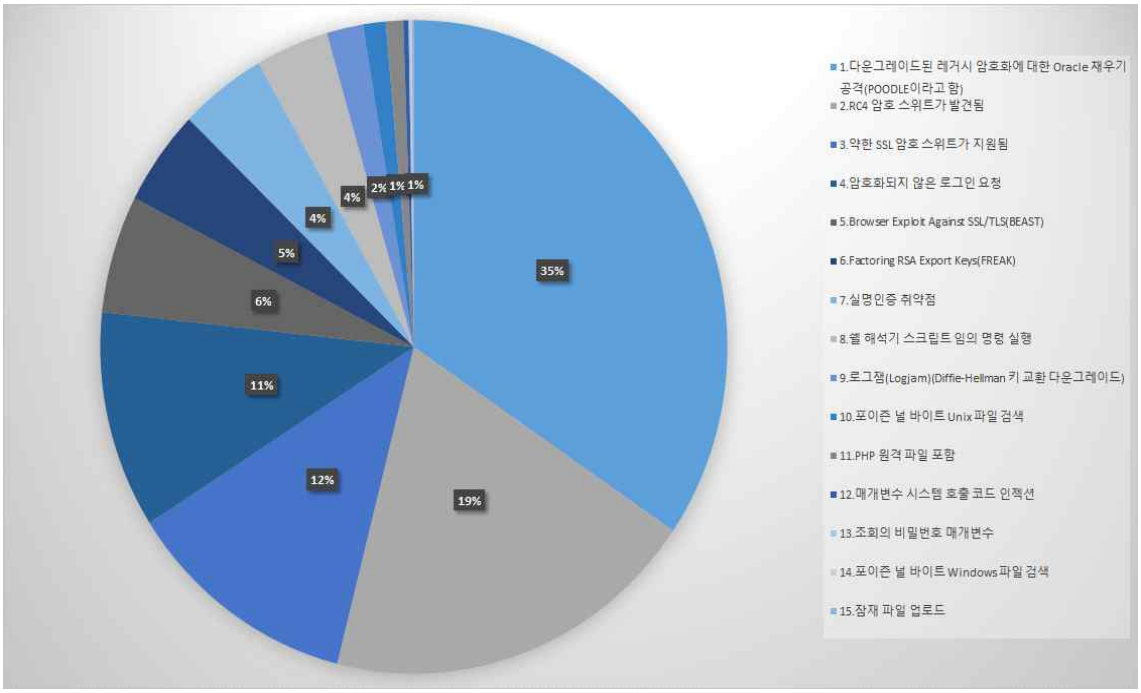
웹 어플리케이션 상에서 모든 실행 경로에 대해서 접근제어를 검사하지 않거나 불완전하게 검사하는 취약점을 이용하여 임의의 명령 실행이 가능한 악의적인 파일을 서버로 업로드하여 권한을 탈취할 수 있는 취약점

17 에러처리 취약점

웹 서버에 별도의 에러페이지를 설정하지 않은 경우, 에러 메시지를 통해 서버 데이터 정보 등 공격에 필요한 정보가 노출되는 취약점

제2절 웹 취약점 유형 별 주요 탐지현황

과학기술사이버안전센터에서는 웹 어플리케이션 분야의 취약점을 탐지하기 위하여 다수의 패턴을 보유하고 있으며, 앞서 분류된 웹 취약점 유형들이 포함하고 있는 주요 탐지패턴 현황은 아래와 같다. 이번 보고서에는 17년도 1/4분기에 주로 탐지된 19개의 취약점 패턴 분석내용을 중점적으로 다루도록 한다.



순번	취약점 유형	주요 탐지패턴
1	실명인증 취약점	[1-1] 실명인증 취약점
2	전송 시 개인정보 노출 취약점	[2-1] 암호화되지 않은 로그인 요청
		[2-2] RC4 암호 스위트가 발견됨
		[2-3] Browser Exploit Against SSL/TLS (BEAST)
		[2-4] 약한 SSL 암호 스위트가 지원됨
		[2-5] Logjam (Diffe-Hellman key exchange downgrade)
		[2-6] Factoring RSA Export Keys (FREAK)
		[2-7] 다운그레이드된 레거시 암호화에 대한 Oracle 채우기 공격 (POODLE)
		[2-8] 조회의 비밀번호 매개변수

순번	취약점 유형	주요 탐지패턴
2	전송 시 개인정보 노출 취약점	[2-9] 더 이상 사용되지 않는 SSL 버전이 지원됨
		[2-10] SHA-1 암호 스위트가 발견됨
		[2-11] 더 이상 사용되지 않는 취약한 eNcryption(a.k.a. DROWN)을 사용하는 RSA 복호화
		[2-12] 암호화되지 않은 _VIEWSTATE 매개변수
3	파일 다운로드 취약점	[3-1] 포이즌 널 바이트 Windows 파일 검색
		[3-2] 포이즌 널 바이트 Unix 파일 검색
4	파일 업로드 취약점	[4-1] 잠재 파일 업로드
		[4-2] PHP 원격 파일 포함
		[4-3] 쉘 해석기 스크립트 임의 명령 실행
		[4-4] 매개변수 시스템 호출 코드 인젝션

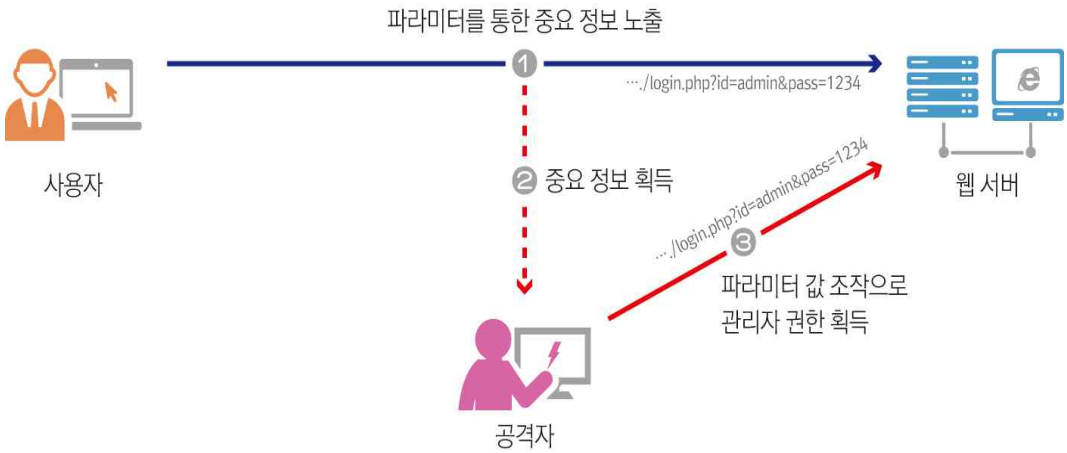
<2017년도 1/4분기 웹 취약점 유형 탐지현황>

제3장 웹 취약점 유형 별 상세 조치방안

제1절 실명인증 취약점

취약점 설명

로그인을 통한 본인 확인 검증절차에서 암호화 체계가 미흡한 경우 사용자 중요 정보가 평문으로 전송될 수 있다. 공격자는 이러한 취약점을 악용하여 개인정보 탈취 시도를 할 수 있으며, 관리자 권한 탈취 시 기밀정보 유출 및 홈페이지 위□변조 등과 같은 2차 공격의 위험이 존재한다.



<계정접속 파라미터 조작을 통한 권한 획득>

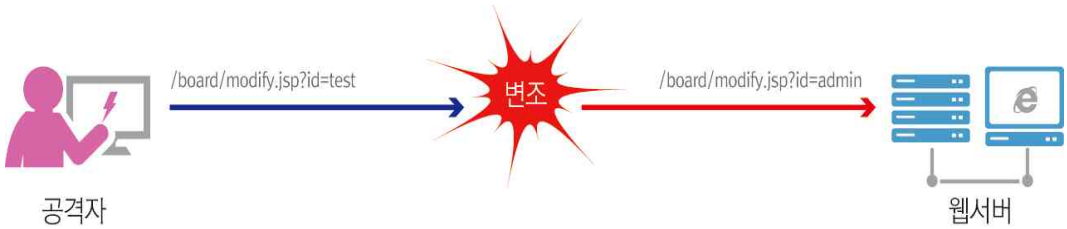
사전예방조치 방안

* 안전한 라이브러리 및 프레임워크로 개발된 프로그램 사용
* 강력한 알고리즘 기반의 암호화 통신
* 사용자들의 인증 값을 게시물 작성 페이지 및 저장 페이지에서 Server Side Script를 통하여 점검 하도록 수정 권장

[1-1] 실명인증 취약점

◎ 개요

웹서버 구성 시 계정 인증 요청 값에 대한 검증 기능이 적용되지 않을 경우, 공격자는 해당 페이지의 URL을 직접 입력하여 강제 접속이나 권한 획득을 통해 중요 정보 유출 또는 서비스의 무단 사용 등의 문제가 발생할 수 있는 취약점



<계정인증 요청 값에 대한 검증체계 미흡>

◎ 조치방안

* 사용자의 권한에 따른 ACL(Access Control List)을 관리
* 인증 프레임워크 사용(JAAS Authorization Framework, OWASP ESAPI Access Control 등)

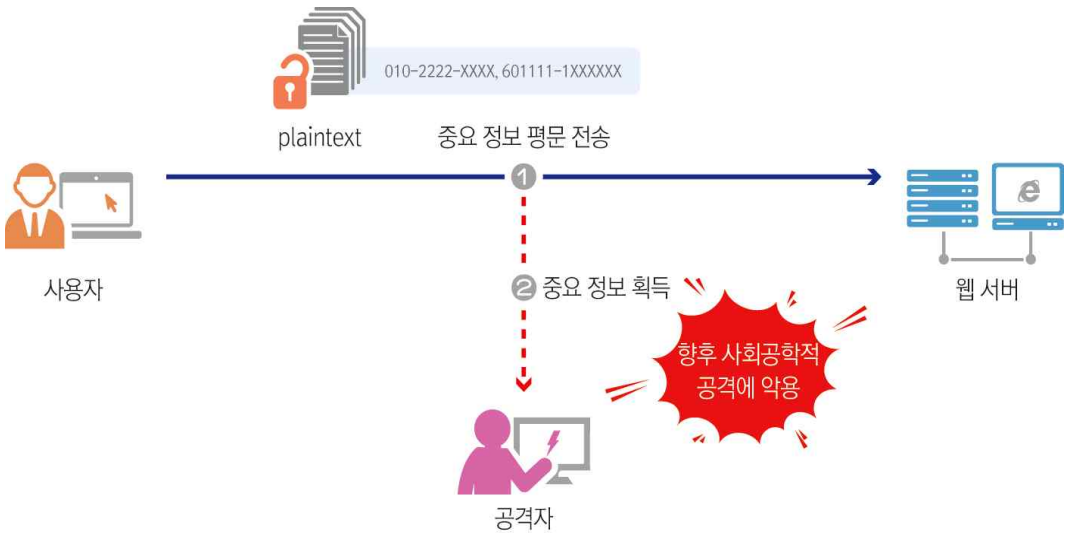
안전한 코드 예제

```
public void f(String sSingleId, int iFlag, String sServiceProvider, String sUid, String sPwd)
{
.....
env.put(Context.PROVIDER_URL, sServiceProvider);
// 익명의 인증을 사용하지 않음
env.put(Context.SECURITY_AUTHENTICATION, “simple”);
env.put(Context.SECURITY_PRINCIPAL, sUid);
env.put(Context.SECURITY_CREDENTIALS, sPwd);
.....
```

제2절 전송 시 개인정보 노출 취약점

취약점 설명

사용자와 서버 간 통신 시, 암호화 처리되지 않은 구간 사이에서 보안과 관련된 민감한 데이터 통신이 평문으로 전송 처리될 경우 통신상의 모든 정보가 스니핑을 통해 인가되지 않은 사용자에게 탈취, 노출, 위□변조될 수 있는 위험성이 존재한다.



<평문으로 노출된 개인정보 탈취>

사전예방조치 방안

* 세션 객체를 사용하여 데이터 위/변조 차단(쿠키는 조작이 가능함)
* SSL/TLS (HTTPS)의 암호화 통신 처리
* Secure Coding

JAVA

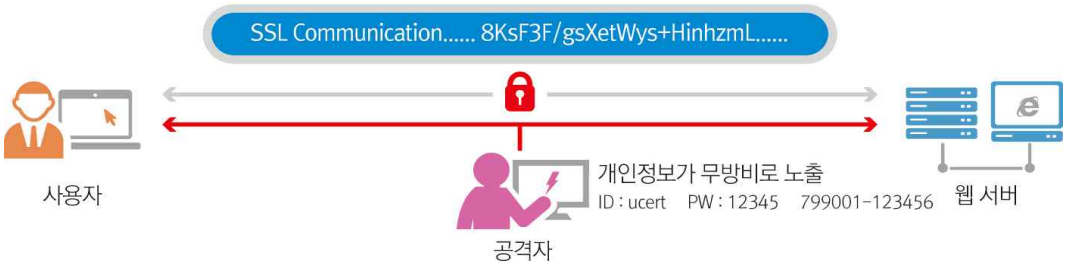
```
Socket s = new Socket(serverAddr,9440);
Cipher crypt = Cipher.getInstance("RSA"); //암호화 통신 처리
...(중략)
encryptedStr=crypt.update(answer.getBytes());
```

[2-1] 암호화되지 않은 로그인 요청

◎ 개요

로그인 또는 실명인증 시 사용자로부터 입력받은 중요 정보가 암호화 되지 않고 평문으로 송수신할 경우 스니핑을 통해 사용자의 중요 정보가 공격자에게 노출될 수 있는 취약점

* 중요정보 예시 : ID/Password, 주민증록번호, 신용카드정보, 여권번호, 이메일 주소, 전화번호 등



<중요정보 평문노출>

◎ 조치방안

* 중요 정보 전송 시 암호화 전송을 위한 HTTPS 프로토콜 사용 권장
* 최소 128비트 길이의 키를 이용한 암호화 권장

중요정보 전송 시 암호화 코드 예시

```
……
String getPassword() {
return "secret_Password";
}

void  foo() {
try {
Socket socket = new Socket("taranis", 4444);
PrintStream out = new PrintStream(socket.getOutputStream(), true);
Cipher c = Cipher.getInstance("RSA"); // 암호화 프로토콜 설정
StringPassword= getPassword();
byte[]encryptedStr = c.update(Password.getBytes());
out.write(encryptedStr, 0, encryptedStr.length);
} catch (FileNotFoundException) {
 ……
}
```

[2-2] RC4 암호 스위트가 발견됨

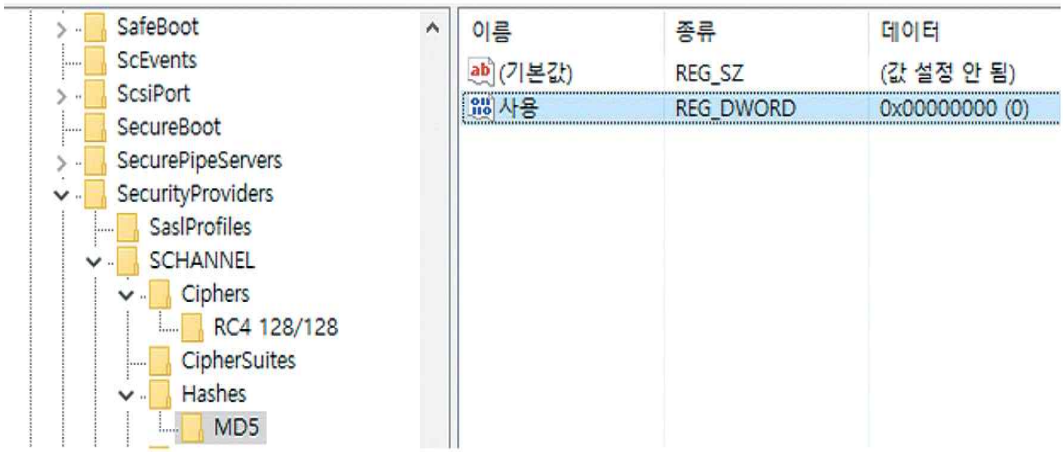
◎ 개요

RC4 암호 스위트는 보안에 취약한 암호화 알고리즘을 사용하는 SSL/TLS 패키지로써, 공격자는 이를 악용하여 사용자의 세션 정보를 불법적으로 탈취 및 변조를 통해 정상적인 사용자로 위장 접근이 가능함

◎ 조치방안

* MD5, RC4와 같은 취약한 알고리즘을 사용하지 못하도록 설정

Windows



<레지스트리 편집기>

<레지스트리 편집>

· HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Control/SecurityProviders /SCHANNEL/Ciphers

· RC4 128/128(새로 작성 › 키 RC4 128/128)이라는 새 키를 작성

· 키의 이름을 마우스 오른쪽 단추로 클릭하고 '사용' 메뉴에서 DWORD(32비트) 값을 작성(새로 작성 › DWORD(32비트) 값 › 사용), 기본값 0으로 설정

· HKEY_LOCAL_MACHINE/SYSTEM/CurrentControlSet/Control/SecurityProviders /SCHANNEL/Hashes

· MD5(새로 작성 › 키 › MD5) 키를 작성

· 키의 이름을 마우스 오른쪽 단추로 클릭하고 '사용' 메뉴에서 DWORD(32비트) 값 작성(새로 작성 › DWORD(32비트) 값 › 사용), 기본값 0으로 설정

· 저장 후 닫기

Apache

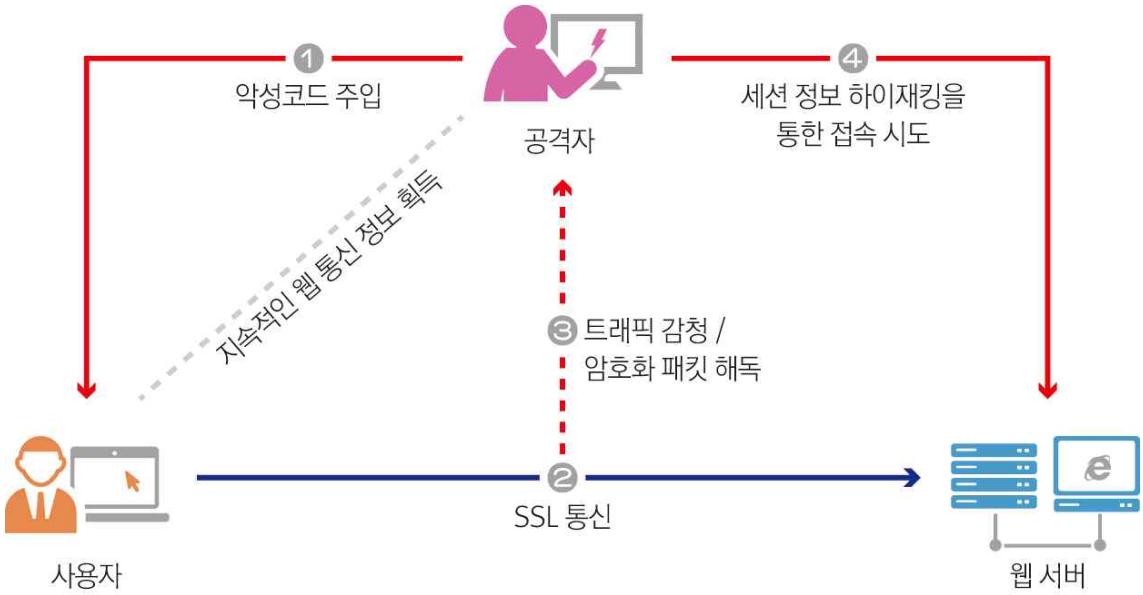
* Conf/extra/http-ssl.conf 파일에서 다음 주석 처리를 통해 취약한 옵션 제거

```
#SSLCipherSuite HIGH:MEDIUM:!MD5:!RC4
SSLCipherSuiteECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCMSHA256:ECD
HE-RSA-AES256-GCMSHA384:₩
ECDHE-ECDSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-DSS-AES128
GCM-SHA256:₩
ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA:₩
ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA384:₩
ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-SHA256:₩
DHE-RSA-AES128-SHA:DHE-DSS-AES128-SHA256:DHE-RSA-AES256-SHA256:₩
DHE-DSS-AES256-SHA:DHE-RSA-AES256-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:₩
AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:AES:CAMELLIA:DES-CBC3-SHA:₩
!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!PSK:!aECDH:!EDH-DSS-DES-CBC3-SHA:₩
!EDH-RSA-DES-CBC3-SHA:!KRB5-DES-CBC3-SHA
#SSLProxyCipherSuite HIGH:MEDIUM:!MD5:!RC4
SSLProxyCipherSuiteECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES128-GCMSHA25
6:ECDHE-RSA-AES256-GCMSHA384:₩
ECDHE-ECDSA-AES256-GCM-SHA384:DHE-RSA-AES128-GCM-SHA256:DHE-DSS-AES128GC
M-SHA256:₩
ECDHE-RSA-AES128-SHA256:ECDHE-ECDSA-AES128-SHA256:ECDHE-RSA-AES128-SHA:₩
ECDHE-ECDSA-AES128-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-ECDSA-AES256-SHA384:₩
ECDHE-RSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA:DHE-RSA-AES128-SHA256:₩
DHE-RSA-AES128-SHA:DHE-DSS-AES128-SHA256:DHE-RSA-AES256-SHA256:₩
DHE-DSS-AES256-SHA:DHE-RSA-AES256-SHA:AES128-GCM-SHA256:AES256-GCM-SHA384:₩
AES128-SHA256:AES256-SHA256:AES128-SHA:AES256-SHA:AES:CAMELLIA:DES-CBC3-SHA:₩
!aNULL:!eNULL:!EXPORT:!DES:!RC4:!MD5:!PSK:!aECDH:!EDH-DSS-DES-CBC3-SHA:₩
!EDH-RSA-DES-CBC3-SHA:!KRB5-DES-CBC3-SHA
#SSLProtocol all -SSLv3
SSLProtocol all -SSLv3 -TLSv1
#SSLProxyProtocol all -SSLv3
SSLProxyProtocol all -SSLv3 -TLSv1
```

[2-3] Browser Exploit Against SSL/TLS(BEAST)

◎ 개요

취약한 버전의 SSL 프로토콜을 사용할 경우, 중간자 공격을 통하여 쿠키 정보 탈취 및 세션 하이재킹이 가능하며, 암호화 데이터를 평문으로 복호화 할 수 있는 취약점



<중간자 공격을 통한 세션정보 탈취>

◎ 조치방안

1 SSL 3.0, TLS 1.0 비활성화

* mod_ssl에서 SSL 3.0 비활성화

· /etc/httpd/conf.d/ssl.conf 경로에 Protocol 지시문 설정

설정 1 SSLv2와 SSLv3을 비활성화(SSLv2와 SSLv3 이외는 모두 사용)

SSLProtocol All -SSLv2 -SSLv3 //SSLv2와 SSLv3을 비활성화

service httpd restart // httpd를 재시작

설정 2 TLSv1.x 제외하고 모두 사용 안 함

SSLProtocol -All +TLSv1 +TLSv1.1 +TLSv1.2 // **Red Hat Enterprise Linux 6.6 또는 7 이후의 경우**

SSLProtocol -All +TLSv1 // **Red Hat Enterprise Linux 5를 포함한 다른 플랫폼의 경우**

service httpd restart // **httpd 재시작**

* mod_nss에서 SSL 3.0 비활성화

· /etc/httpd/conf.d/nss.conf 경로에 NSSProtocol 지시문 설정

NSSProtocol TLSv1.0, TLSv1.1 // **Red Hat Enterprise Linux 6 이상인 경우**

NSSProtocol TLSv1.0 // **Red Hat Enterprise Linux 5인 경우**

service httpd restart // **httpd 재시작**

2 브라우저 버전 업그레이드(TLS 1.1, 1.2사용 가능한 버전)

* '크롬' 16 이상, 'MS' MS12-006 패치, 'Firefox' 10 이상으로 업그레이드 권장

3 TLS 1.1, 1.2 버전 패치 및 프로토콜 설정

Tomcat /conf/server.xml 설정

```
<Connector port="8443" Protocol="HTTP/1.1" SSLEnabled="true"
.....
  clientAuth="false"
  sslEnabledprotocols="TLSv1.2"
  sslProtocol="TLSv1.2"
  keystorePass=키스토어패스워드
  keystoreFile="conf/test/test.keystore"
  ciphers="TLS_RSA_WITH_AES_256_CBC_SHA256,
TLS_DH_DSS_WITH_AES_256_CBC_SHA256,TSL_DH_RSA_WITH_AES_256_CBC_SHA_256,
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256,TLS_DH_anon_WITH_AES_256_CBC_SHA256"
```

[2-4] 약한 SSL 암호 스위트가 지원됨

◎ 개요

SSL 구성 시 취약한 암호화 알고리즘으로 구성된 모듈을 사용할 경우, 암호 데이터가 쉽게 평문으로 복호화될 수 있는 위험이 있음

◎ 조치방안

* 웹서버 보안설정

· Tomcat : /conf/server.xml

· JBOSS : /server/all/deploy/jbossweb.sar/server.xml

안전한 설정 예시

```
<Connector protocol="HTTP/1.1" SSLEnabled="true"
  port="443" address="${jboss.bind.adress}"
  scheme="https" secure="true" clientAuth="false"
  keystoreFile="${jboss.sever.home.dir}/conf/test.keystore"
  keystorePass="Password"
  sslProtocol = "TLS" ciphers="SSL_RSA_WITH_RC4_128_MD5,
SSL_RSA_WITH_RC4_128_SHA, TLS_RSA_WITH_AES_128_CBC_SHA,
TLS_DHE_RSA_WITH_AES_128_CBC_SHA, TLS_DHE_DSS_WITH_AES_128_CBC_SHA,
SSL_RSA_WITH_3DES_EDE_CBC_SHA, SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA,
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA"/>

//설정 후 JBoss 또는 Tomcat 서버 재기동
```

[2-5] Logjam(Diffie-Hellman key exchange downgrade)

◎ 개요

Logjam은 TLS의 설계상의 취약점으로 발생하며, 웹 또는 이메일 서버 간 취약한 암호화 기법을 사용할 경우 중간자 공격을 통해 HTTPS 연결에서 OpenSSL을 포함한 SSL/TLS 클라이언트를 다운그레이드 시킬 수 있는 취약점

◎ 조치방안

[1] 관리자는 SSL를 사용하는 경우, OpenSSL의 최신 버전을 실행하고 있는지 확인하여야 하며, 아래와 같이 해당 버전의 SSL 버전을 업그레이드하여야 함

* OpenSSL 1.0.2 : 패치된 버전 1.0.2b 이상
* OpenSSL 1.0.1 : 패치된 버전 1.0.1n 이상

[2] OpenSSL 버전 확인방법 예시

```
$ openssl version
  OpenSSL 1.0.2j 26 Sep 2016
```

[2-6] Factoring RSA Export Keys(FREAK)

◎ 개요

웹서버와 클라이언트 간 암호화 통신을 위해 사용되는 RSA 암호화키를 해독하기 쉬운 수출용 RSA 암호화키로 다운그레이드 시키는 취약점으로, 이것을 악용하여 중간자 공격(MITM)을 통해 암호화 통신을 해독하거나 또 다른 악성 코드를 주입시키는 일이 가능함

◎ 조치방안

[1] 취약한 버전의 SSL 프로토콜 및 브라우저 업그레이드

* OpenSSL 0.9.8 사용자는 OpenSSL 0.9.8zd로 업그레이드
* OpenSSL1.0.0 사용자는 OpenSSL1.0.0p로 업그레이드
* OpenSSL1.0.1 사용자는 OpenSSL1.0.1k로 업그레이드

PC 브라우저

* 윈도우 사용자는 윈도우 보안 업데이트
* 크롬 사용자는 41버전 이상으로 업그레이드

2 OpenSSL 설정 비활성화

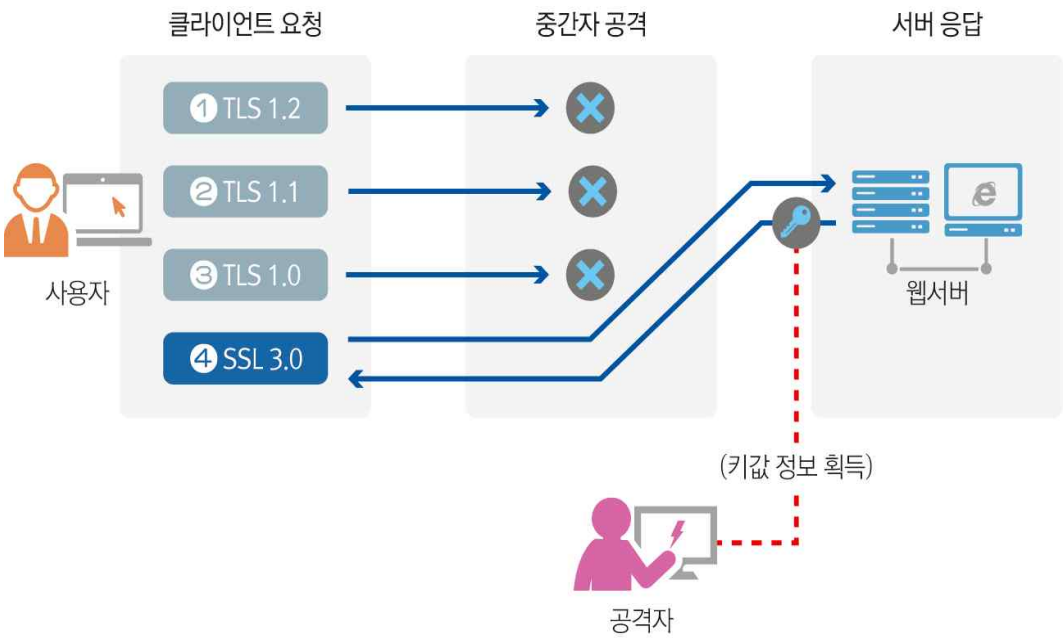
Apache

```
RSA_EXPORT suite 비활성화 (!EXPORT 추가 설정) /conf/extra/httpd-ssl.conf 파일
SSLProtocol ALL -SSLv2 -SSLv3
SSLCipherSuite ALL:!aNULL:!eNULL:!EXPORT:!PSK:!MD5:!RC4:!DES
Listen 443
AddType application/x-x509-ca-cert .crt AddType
application/x-pkcs7-crl .crl
```

[2-7] 다운그레이드된 레거시 암호화에 대한 Oracle 채우기 공격(POODLE)

◎ 개요

공격자가 클라이언트와 서버가 SSL 통신을 맺기 위한 과정에서 개입하여 취약한 암호화 프로토콜인 SSL 3.0으로 강제 다운그레이드를 수행하는 중간자 공격을(MITM) 통해 민감한 사용자 데이터를 탈취할 수 있는 취약점



<취약한 암호화 프로토콜 환경에서의 중간자 공격>

◎ 조치방안

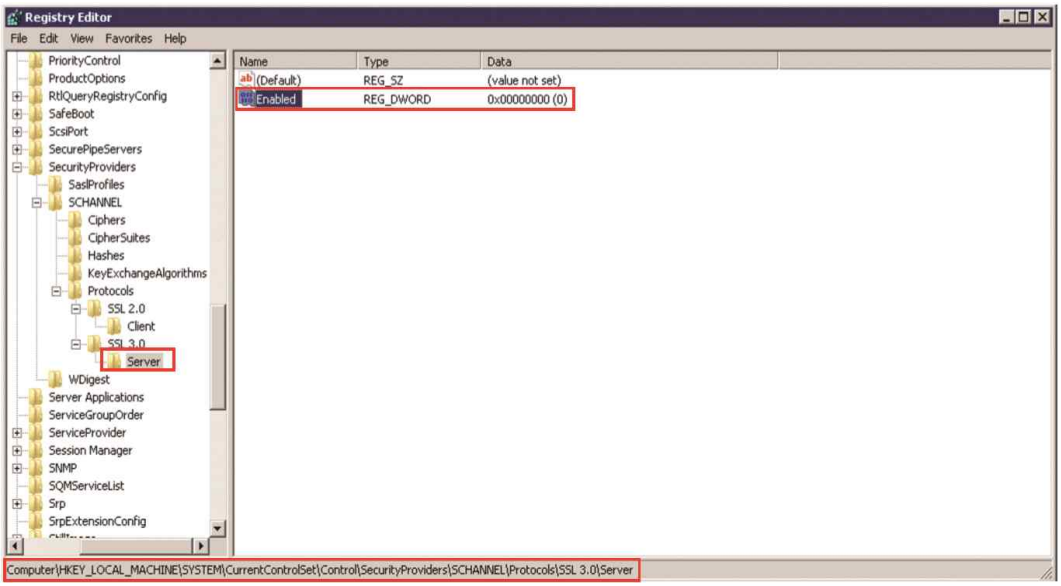
* SSLv3.0 비활성화 설정
* TLS1.0 이상의 버전 권장

Apache

/etc/apache2/mods-available/ssl.conf
SSLProtocol ALL -SSLv2 -SSLv3 # SSLv2,v3 비활성화

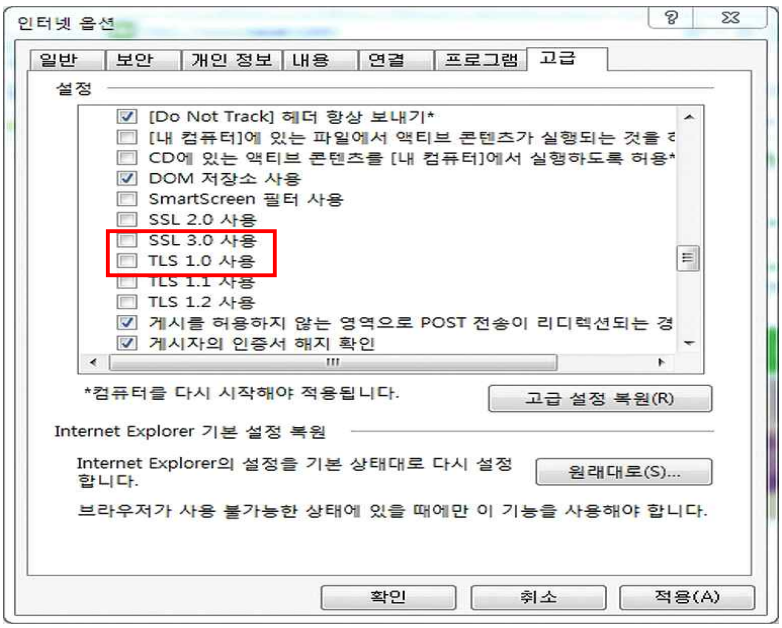
IIS

레지스트리 위치 :
HKEY_LOCAL_MACHINE/Comm/SecurityProviders/SCHANNEL/Protocols/SSL 3.0/Server
DWORD 이름 : 사용
DWORD 값 : 0



<SSL 3.0 레지스트리 비활성화 설정>

클라이언트



<SSL 3.0 비활성화 설정>

[2-8] 조회의 비밀번호 매개변수

◎ 개요

계정정보, 콘텐츠 정보 등 정보조회에 활용되는 URL 매개변수 값에 사용자명 및 ID/Password와 같은 민감한 계정 정보가 암호화 되지 않은 상태로 전송 값이 노출되는 취약점

◎ 조치방안

* 중요 정보 전송 구간 암호화 조치
  - 사용자명, Password
  - 주민등록번호, 신용카드번호
  - 이메일 주소, 전화번호

[2-9] 더 이상 사용되지 않는 SSL 버전이 지원됨

◎ 개요

취약한 버전의 SSL 프로토콜을 사용할 경우, 공격자가 클라이언트와 서버 간의 암호화 데이터를 평문으로 복호화하는 중간자 공격을 실행하여 중요한 정보를 열람할 수 있는 취약점이다.

◎ 조치방안

1 SSLv1, SSLv2, SSLv3 비활성화(disable)

2 TLS 사용 권장

Apache

```
/conf/server.xml 설정
  〈Connector port="443" protocol="org.apache.coyote.http11.Http11Protocol" SSLEnabled="true"
  maxThreads="150" minSpareThreads="25" maxSpareThreads="75" enableLookups="false“
  ... (생략)
  useServerCipherSuitesOrder="true" clientAuth="false"
  sslProtocol="TLS" sslEnabledProtocols="TLSv1, TLSv1.1, TLSv1.2“
  ....(생략)" /〉
```

* 중요정보가 외부에 탈취되는 것을 방지하기 위해 강력한 암호화 프로토콜 권장

[2-10] SHA-1 암호 스위트가 발견됨

◎ 개요

암호화 스위트(Cipher Suite)는 SSL/TLS 암호화 통신을 위해 사용되는 암호화 알고리즘 패키지로 서버와 클라이언트의 암호화 과정에서 사용할 프로토콜 및 암호화통신 정보를 담고 있다. 이를 구성하는 암호화 알고리즘이 취약하게 설정되어 있을 경우, 암호 데이터가 쉽게 평문으로 복호화 될 수 있는 위험성이 있으며 외부 공격자는 이를 악용하여 디지털 서명 및 파일등의 무결성을 깨트릴 수 있다.

<취약한 SHA-1알고리즘 사용 시 발생할 수 있는 해시충돌 공격>

◎ 조치방안

1 Apache 암호화 스위트 설정

```
<Connector port="443" protocol="org.apache.coyote.http11.Http11Protocol" SSLEnabled="true"
maxThreads="150" minSpareThreads="25" maxSpareThreads="75" enableLookups="false"
… (생략)
Ciphers="ECDHE-ECDSA-AES256-GCM-SHA384, ECDHE-RSA-AES256-GCM-SHA384,
         ECDHE-ECDSA-CHACHA20-POLY1305, ECDHE-RSA-CHACHA20-POLY1305,
         ECDHE-ECDSA-AES128-GCM-SHA256, ECDHE-RSA-AES128-GCM-SHA256,
         ECDHE-ECDSA-AES256-SHA384, ECDHE-RSA-AES256-SHA384,
         ECDHE-ECDSA-AES128-SHA256, ECDHE-RSA-AES128-SHA256" />
```

* SHA-1과 같은 취약한 알고리즘을 사용하지 않을 시 암호 데이터가 평문으로 복호화 될 수 있기 때문에 권장하는 암호화 알고리즘으로 변경

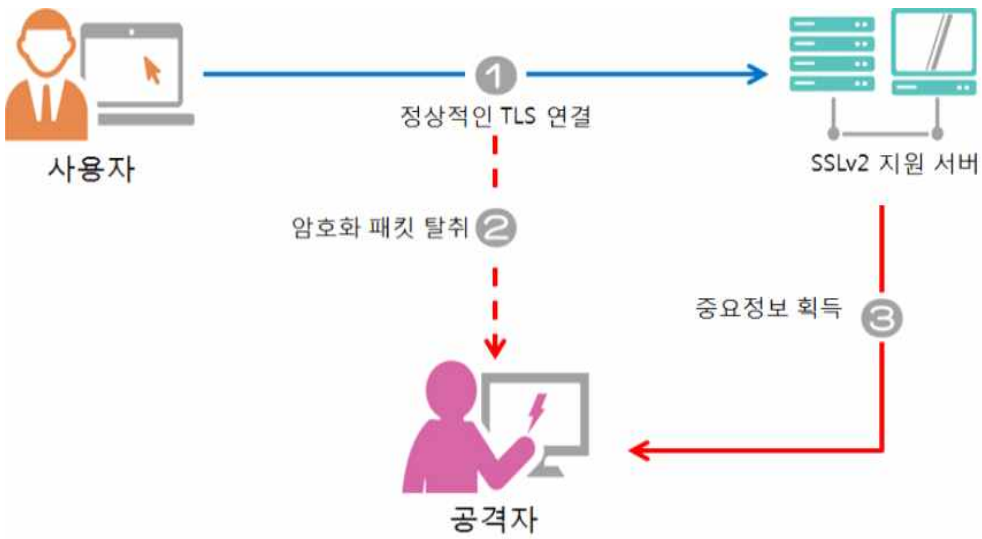
[2] 권장되는 암호화 스위트 적용

```
0xC0,0x2C  -  ECDHE-ECDSA-AES256-GCM-SHA384   TLSv1.2  Kx=ECDH  Au=ECDSA  Enc=AESGCM(256)     Mac=AEAD
0xC0,0x30  -  ECDHE-RSA-AES256-GCM-SHA384     TLSv1.2  Kx=ECDH  Au=RSA    Enc=AESGCM(256)     Mac=AEAD
0xCC,0x14  -  ECDHE-ECDSA-CHACHA20-POLY1305   TLSv1.2  Kx=ECDH  Au=ECDSA  Enc=ChaCha20(256)  Mac=AEAD
0xCC,0x13  -  ECDHE-RSA-CHACHA20-POLY1305     TLSv1.2  Kx=ECDH  Au=RSA    Enc=ChaCha20(256)  Mac=AEAD
0xC0,0x2B  -  ECDHE-ECDSA-AES128-GCM-SHA256   TLSv1.2  Kx=ECDH  Au=ECDSA  Enc=AESGCM(128)     Mac=AEAD
0xC0,0x2F  -  ECDHE-RSA-AES128-GCM-SHA256     TLSv1.2  Kx=ECDH  Au=RSA    Enc=AESGCM(128)     Mac=AEAD
0xC0,0x24  -  ECDHE-ECDSA-AES256-SHA384       TLSv1.2  Kx=ECDH  Au=ECDSA  Enc=AES(256)       Mac=SHA384
0xC0,0x28  -  ECDHE-RSA-AES256-SHA384         TLSv1.2  Kx=ECDH  Au=RSA    Enc=AES(256)       Mac=SHA384
0xC0,0x23  -  ECDHE-ECDSA-AES128-SHA256       TLSv1.2  Kx=ECDH  Au=ECDSA  Enc=AES(128)       Mac=SHA256
0xC0,0x27  -  ECDHE-RSA-AES128-SHA256         TLSv1.2  Kx=ECDH  Au=RSA    Enc=AES(128)       Mac=SHA256
```

[2-11] 더 이상 사용되지 않는 취약한 eNcryption(a.k.a. DROWN)을 사용하는 RSA 복호화 중

◎ 개요

최신 서버와 클라이언트들은 TLS 프로토콜을 통한 암호화 통신을 사용한다. 그러나 현재까지도 암호화 데이터의 복호화가 가능한 취약점이 존재하는 SSLv2를 사용하는 서버들이 다수 있다. 공격자는 이러한 SSLv2 취약점을 악용하여 TLS 통신을 가로채는 중간자 공격(MITM)을 수행 후 HTTPS 연결을 복호화 하여 비밀번호, 신용카드 번호나 재무관련 데이터들과 같은 민감한 통신데이터를 획득할 수 있다.



<SSLv2 사용 시 중간자 공격을 통한 중요정보 획득>

◎ 조치방안

1 OpenSSL 최신 버전 업그레이드

CentOS, Redhat

```
# yum clean
# yum update openssl
```

Ubuntu

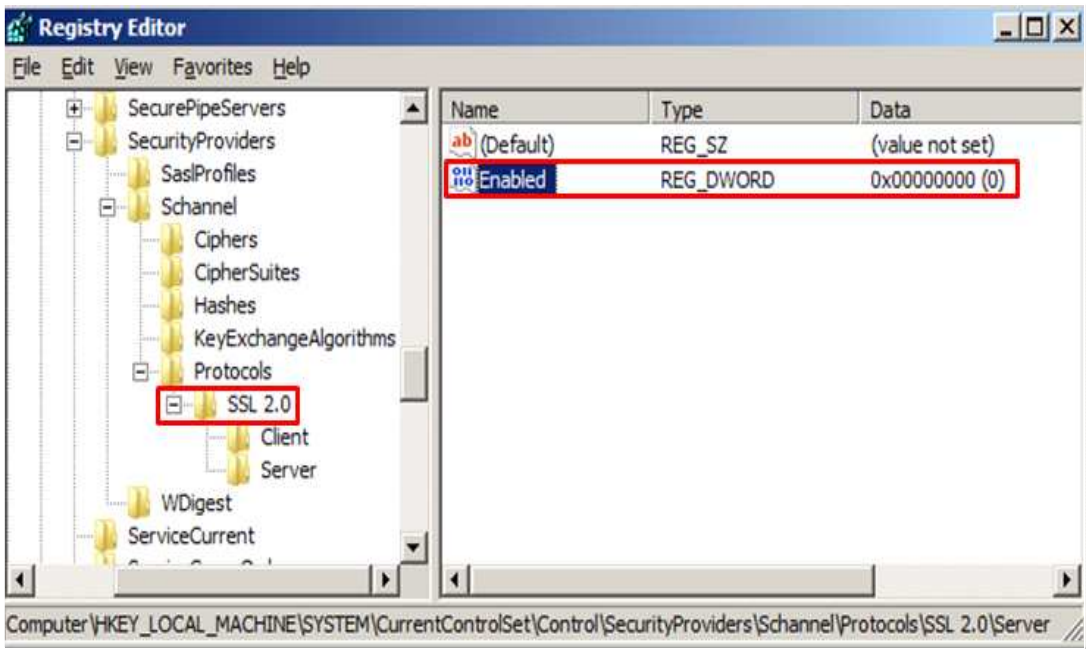
```
# apt-get upgrade openssl
```

* OpenSSL 최신 버전으로 업그레이드 및 SSLv2 비활성화 설정

2 SSLv2 비활성화 설정

IIS

경로 : HKey_Local_Machine₩System₩CurrentControlSet₩Control₩SecurityProviders₩SCHANNEL₩Protocols
DWORD 이름 : 사용
DWORD 값 : 0



<SSL 2.0 비활성화 레지스트리 수정>

Apache

```
# vi httpd.conf
.....
SSLProtocol -all +TLSv1.1 +TLSv1.2
SSLCipherSuite HIGH:!aNULL:!MD5:!SSLv2:!SSLv3:!TLSv1
.....
# service httpd restart
```

nginx

```
ssl_protocols TLSv1.1 TLSv1.2;
nigx restart
```

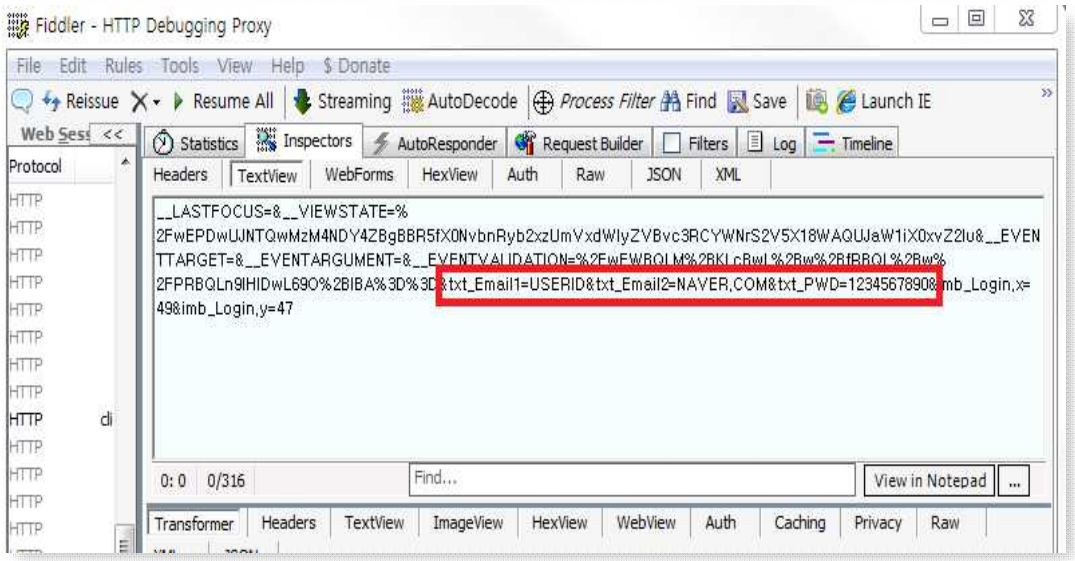
Postfix

```
smtpd_tls_protocols = !SSLv2, !SSLv3
smtpd_tls_mandatory_protocols = !SSLv2, !SSLv3
tlsproxy_tls_protocols = $smtpd_tls_protocols
tlsproxy_tls_mandatory_protocols = $smtpd_tls_mandatory_protocols
smtp_tls_protocols = !SSLv2, !SSLv3smtp_tls_mandatory_protocols = !SSLv2, !SSLv3
lmtp_tls_protocols = !SSLv2, !SSLv3
lmtp_tls_mandatory_protocols = !SSLv2, !SSLv3
smtpd_tls_ciphers = medium
smtp_tls_ciphers = medium
smtpd_tls_dh1024_param_file=${config_directory}/dh2048.pem
smtpd_tls_eecdh_grade = strong
smtpd_tls_exclude_ciphers =
EXPORT, LOW, MD5, SEED, IDEA, RC2
smtp_tls_exclude_ciphers =
EXPORT, LOW, MD5, aDSS, kECDHe, kECDHr, kDHd, kDHr, SEED, IDEA, RC2
```

[2-12] 암호화되지 않은 _VIEWSTATE 매개변수

◎ 개요

ViewState는 ASP.NET에서 생성한 HTML 페이지에서 사용되는 페이지 상태를 관리하는데, 페이지 실행 시 수집되는 값을 Viewstate 값에 할당하여 클라이언트 브라우저에 일시적으로 저장된 상태로 클라이언트가 해당 페이지를 서버에 다시 게시하도록 선택할 경우(postback) 숨겨진 필드에 포함된 평문정보를 볼 수 있는 취약점이 존재한다. 따라서 Viewstate 매개변수를 암호화 하지 않을 경우 이를 악용하는 공격자에게 사용자 개인정보 및 웹 어플리케이션에 대한 구성정보가 평문 형태로 노출 될 수 있다.



<VIEWSTATE 매개변수 암호화 미적용 시 중요정보 평문 노출>

◎ 조치방안

* 정보를 평문으로 볼 수 없게 암호화 알고리즘 설정

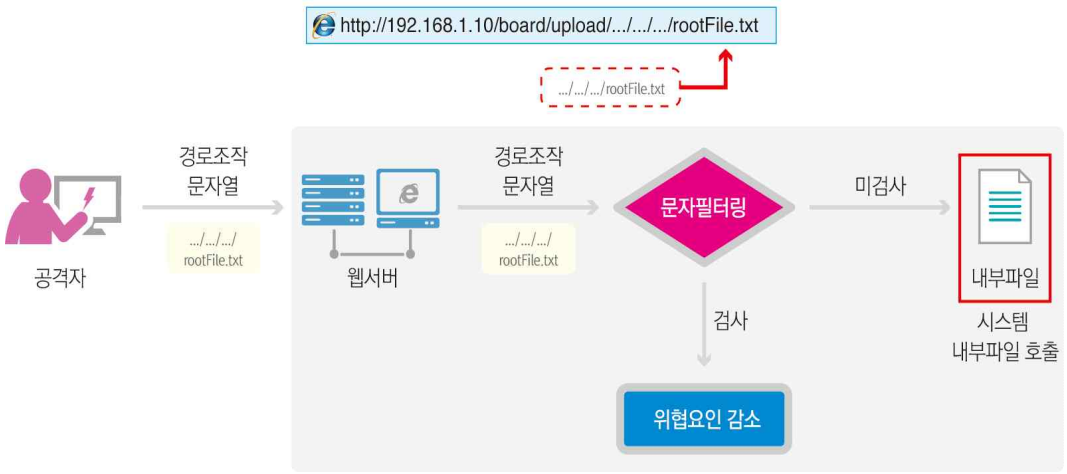
· <system.web> 요소 아래 Web.Config 파일 암호화 알고리즘 설정 예시

```
〈configuration〉
    〈system.web〉
        〈machineKey validation = "3DES"/〉
    〈/system.web〉
〈/ configuration〉
```

제3절 파일 다운로드 취약점

취약점 설명

외부에서 입력되는 값에 대해 경로 조작에 사용될 수 있는 문자(../, %2e%2e%ef 등)를 검증하지 않고, 주요 시스템 파일이 저장된 경로에 접근 권한이 설정되어 있지 않을 경우, 공격자가 임의의 위치에 있는 파일의 다운로드를 할 수 있으며, 다운로드 된 시스템 파일 정보 획득을 통해 2차 공격에 악용할 수 있다. 특히 유닉스나 리눅스 계열의 웹서버는 각별한 주의가 필요하다.



<경로조작을 통한 시스템 내부파일 다운로드>

사전예방조치 방안

* 다운로드 파일 이름을 DB에 저장하고 다운로드 시 요청파일 이름과 비교하여 적정성 검증을 통한 취약한 매개변수 제거
* 파일 경로 파라미터 변수에서 상위 경로를 의미하는 path traversal 문자열 필터링 (필터링할 문자열 : [..], [../], [..₩])
* 다운로드가 가능한 위치는 고정적으로 설정, 그 외 디렉터리에서는 다운로드 불가능하도록 로직 구현

* Secure Coding

ASP에서 필터링 구현예시

```
<%
 file = Request.Form ("file")'파일 이름
 Response.ContentType = "application/unknown"'ContentType 선언
 Response.AddHeader "Content-Disposition","attachment; filename=" & file
 Set objStream = Server.CreateObject("ADODB.Stream")'Stream 이용
 strFile = Server.MapPath("./upfiles/") & "₩" & file '서버 절대경로
 strFname=Mid(Fname,InstrRev(file,"₩")+1) '파일 이름 추출, ..₩ 등의 하위경로 탐색은 제거됨
 strFPath = Server.MapPath("./upfiles/") & "₩" & strFname '웹서버의 파일 다운로드 절대경로
 If strFile = strFPath Then'사용자가 다운받는 파일과 웹서버의 파일 다운로드 경로가 맞는지 비교
 objStream.Open objStream.Type = 1
 objStream.LoadFromFile strFile
 download = objStream.Read
 Response.BinaryWrite download
 End If
 Set objstream = nothing'객체 초기화
 %>
```

PHP에서 필터링 구현예시

```
if (preg_match("/[^a-z0-9_-]/i",$up_dir))
print "디렉터리에 특수문자 체크";
exit;
if (preg_match("/[^₩xA1-₩xFEa-z0-9._-]|₩.₩./i",urldecode($dn_file_name)))
print "파일이름에 특수문자 체크";
exit;
$dn_path = "/var/www/data/$up_dir/$dn_file_name";
if (!file_exists($dn_path))
print "파일 존재여부 체크";
exit;
//파일전송 루틴
header("Content-Type: doesn/matter");
header("Content-Length: ".filesize("$dn_path"));
header("Content-Disposition: filename=".$dn_file_name]);
header("Content-Transfer-Encoding:  binary₩r₩n");
header("Pragma: no-cache");
header("Expires: 0");
```

JSP에서 필터링 구현예시

```
<%
// 다운로드 허용할 디렉터리 설정
 String allowPath = "/server/upfiles/";

// 파라미터로부터 다운로드 할 파일을 입력받아 전체경로를 생성함
 String inputPath = getParameter("downfile");
 String filePath = allowPath + inputPath;

// 금지 문자열 리스트
 String blockchar[] = {"..",  "../",      "..₩₩"};

Boolean checkResult = true;

// 금지할 문자열 포함여부 체크
 for(int i=0; i<blockchar.length;i++)  {
 if(filePath.indexOf(blockchar[i]) != -1){
 checkResult = false;
 }
 }

// 파일 전체경로로부터 디렉터리 경로만 추출한 후, 다운로드 허용된 디렉터리인지 체크
 String path = filePath.substring (0, filePath.lastIndexOf("/")+ 1).toLowerCase();
 if(!path.equals(allowPath)  ){
            checkResult = false;
}

if(checkResult == false){
 out.println ("다운로드가 금지된 파일 입니다.");
 }else{
 out.println ("다운로드가 허용된 파일 입니다.");
 }
%>
```

[3-1] 포이즌 널 바이트 Windows 파일 검색

◎ 개요

공격자가 Windows 웹 애플리케이션(ASP)의 콘텐츠 형식에 대한 유효성 검증이 되지 않는 취약점을 악용하여 null 바이트를 사용할 경우 NET API에서 오동작으로 인해 내부정보가 포함된 파일이 다운로드 되는데, 이러한 정보를 활용하여 임의의 위치에 있는 주요 파일 열람 및 다운로드가 가능한 취약점

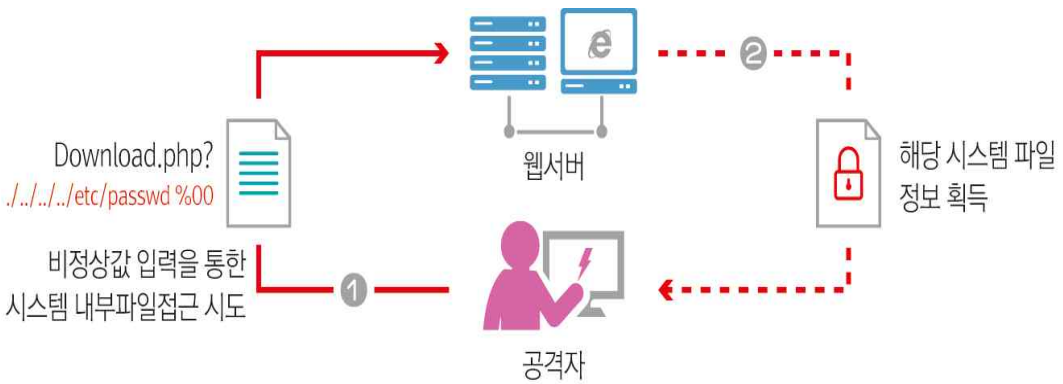
◎ 조치방안

* 파일 다운로드에 사용되는 API를 통해 null 바이트로 입력되는 모든 문자열의 검증기능을 적용하고 비인가자가 임의의 위치에 있는 파일을 열람하거나 다운받지 못하도록 권한을 분리하여야 함
* .NET FrameWork version 1.1 및 SP1, .NET FrameWork version는 null 바이트가 제거된 닷넷 버전 업그레이드 또는 보안패치 적용 수행

[3-2] 포이즌 널 바이트 Unix 파일 검색

◎ 개요

Unix 기반 특정 웹 애플리케이션에서 콘텐츠 형식에 대한 유효성 검증이 되지 않는 취약점을 악용하여 null 바이트를 사용할 경우 오동작으로 인해 내부정보가 포함된 파일이 다운로드 되며, 이러한 정보를 활용하여 임의의 위치에 있는 주요 파일 열람 및 다운로드가 가능한 취약점



<널 바이트 구문을 활용한 시스템 파일 다운로드 시도>

◎ 조치방안

* 소스 내에서 입력된 데이터에 대하여 검증기능 적용

PHP

```
<?php
$clean = str_replace("₩x00", '', $input);
$clean = str_replace("₩0", '', $input);
$clean = str_replace(chr(0), '', $input);
?>
```

JAVA

```
ServletContext.getResource(또는 ServletContext.getResourceAsStream)
ServletContext.getRealPath

ServletContext context = config.getServeletcontext();
String fileName = context.getRealPath(inputPath);
FILE myFile = new File(fileName);
```

제4절 파일 업로드 취약점

취약점 설명

첨부 파일 업로드 기능이 있는 게시판이나 자료실에서 실행파일과 같이 허용되지 않은 확장자에 대한 검증체계가 미흡할 경우, 공격자는 웹쉘과 같은 악성 프로그램 파일을 정상적인 파일로 위장하여 업로드를 수행할 수 있다. 공격자가 악성 프로그램 업로드 후 해당 파일에 접근하여 스크립트를 실행하게 되면 외부 보안체계를 우회하여 내부 정보 획득 및 백도어 설치, 악성파일 삽입 등의 추가적인 공격 수행을 통해 웹서버 장악이 가능하다.

<취약한 게시판을 악용한 웹쉘 업로드>

사전예방조치 방안

* Whitelist 기반 업로드 확장자 검증 기능 구현한 매개변수 제거
* Upload 파일에 대한 실행권한 제거
  · Upload 파일을 위한 전용 디렉터리를 별도 생성하며, 해당 디렉터리에 대한 실행권한 제거

Apache

· FileMatch 지시자를 이용하여 *.ph, *.inc, *lib 등의 Server Side Script 파일에 대해서 직접 URL 호출을 금지
· AddType 지시자를 이용하여 Server Side Script가 실행되지 않도록 설정
· 파일 업로드 디렉터리에 .htaccess 생성 후 아래와 같이 설정

```
.htaccess
<FilesMatch "₩.(ph|inc|lib)">
 Order allow, deny
 Deny from all
 </FilesMatch>
 AddType text/html .html .htm .php .php3 .php4 .phtml .phps .in .cgi .pl .shtml .ns
```

· Apache 설정 파일인 httpd.conf에 해당 디렉터리에 대한 문서 타입을 컨트롤하기 위해 Directory 섹션의 AllowOverride 지시자에서 FileInfo또는 All을 추가

```
<Directory "/usr/local/apache">
AllowOverride FileInfo (또는 All) …….
……..
</Directory>
```

* Secure Coding

ASP에서 확장자 검증 코드의 구현 예시

```
<%
Set Up=Server.CreateObject("SiteGalaxyUpload.Form")
Path1=server.mappath(".") & "₩upload₩"
Fname=Up("file1")
if Fname <> "" then // 파일 첨부가 되었으면
if Up("file1").Size > 10240 then // 용량 제한
Response.Write "용량 초과"
Response.End
end if
if Up("file1").MimeType <> "image" then // 이미지만 업로드 허용
Response.Write "이미지 파일이 아닙니다."
Response.End
end if
Filename=Mid(Fname,InstrRev(Fname,"₩")+1) // 파일이름부분 추출
// 중복 시 파일이름 부분을 변경하기 위해 분리
```

```
Farry=split(Filename,".") // .을 기준으로 분리
preFname=Farry(0)// 파일이름 앞부분
extFname=Farry(1)// 파일의 확장자
// 저장할 전체 path를 만든다, 파일이름을 구한다
 Path2=Path1 & Filename
 saveFname=preFname & "." & extFname
 Set fso=CreateObject("Scripting.FileSystemObject")
 countNo=0 // 파일 중복될 경우 세팅 값
 fExist=0 // 같은 이름의 파일존재 체크
 Do until fExist=1
 If(fso.FileExists(Path2)) Then
 countNo=countNo + 1
 Path2=Path1 & preFname & countNo & "." & extFname
 saveFname=preFname & countNo & "." & extFname
 else
 fExist=1
 End If
 Loop
 Up("file1").SaveAs(Path2)
 response.write(saveFname & "저장완료")
 else
 response.write("Error")
 end if
 Set Up=nothing
 %>
```

PHP에서 확장자 검증 코드의 구현 예시

```
<?php
 $uploaddir='/var/www/uploads/';
//파일사이즈가 0byte 보다 작거나 최대 업로드 사이즈 보다 크면 업로드 금지
if($_FILES['userfile']['name'])
if($_FILES['userfile']['size'] <=0) // 최대 업로드 사이즈 체크 삽입
 print "파일 업로드 에러";
 exit;
//파일 이름의 특수문자가 있을 경우 업로드를 금지
if(eregi("[^a-z0-9₩._₩-]",$_FILES['userfile']['name']))
 print "파일 이름의 특수문자 체크";
 exit;
 //파일 확장자 중 업로드를 허용할 확장자를 정의
```

```
 $full_filename=explode(".", $_FILES['userfile']['name']);
 $extension=$full_filename[sizeof($full_filename)-1];
//Whitelist 방식 확장자 체크
 $extension=strtolower($extension);
 if(!(ereg($extension","hwp") || ereg($extension","pdf") || ereg($extension","jpg")))
 print "업로드 금지파일 입니다";
 exit;
 $uploadfile=$uploaddir.$_FILES['userfile']['name'];
 if(move_uploaded_file($_FILES['userfile']['tmp_name'], $uploadfile))
 print "파일이 존재하고, 성공적으로 업로드 되었습니다.";
 print_r($_FILES);
 else
 print "파일 업로드 공격의 가능성이 있습니다! 디버깅 정보입니다:₩n";
 print_r($_FILES);
 ?>
```

JSP에서 확장자 검증 코드의 구현 예시

```
<%@ page contentType="text/html;charset=euc-kr" %>
<%@ page import="com.oreilly.servlet.MultipartRequest,com.oreilly.servlet.multipart.
 DefaultFileRenamePolicy, java.util.*"%>
<%
 String savePath="/var/www/uploads"; // 업로드 디렉터리
 int sizeLimit=5*1024*1024; // 업로드 파일 사이즈 제한
 try
 MultipartRequest multi=new MultipartRequest(request, savePath,sizeLimit,
 "euc-kr",new DefaultFileRenamePolicy());
 Enumeration formNames=multi.getFileNames(); // 폼의 이름 반환
 String formName=(String)formNames.nextElement();
 String fileName=multi.getFilesystemName(formName); // 파일의 이름 얻기
 String file_ext=fileName.substring(fileName.lastIndexOf('.') + 1);
 if(!(file_ext.equalsIgnoreCase("hwp") || file_ext.equalsIgnoreCase("pdf") ||
 file_ext.equalsIgnoreCase("jpg")) )
 out.print("업로드 금지 파일");
 if(fileName == null)
 out.print("파일 업로드 실패");
 else fileName=new String(fileName.getBytes("8859_1"),"euc-kr"); // 한글인코딩
 out.print("File Name : "+fileName);₩
 catch(Exception e)
```

[4-1] 잠재 파일 업로드

◎ 개요

공격자가 외부로 노출된 게시판의 첨부파일 업로드 기능을 악용하여 악성 스크립트가 포함된 비정상적인 파일 업로드를 수행할 경우, 시스템 장악을 통해 원격에서 시스템 제어 및 위·변조 등 2차 공격에 노출될 수 있는 위험성이 존재함

◎ 조치방안

* **4절 업로드 취약점 사전예방 조치방안** 내용 참고

[4-2] PHP 원격 파일 포함

◎ 개요

PHP에서 "is_a()" 함수가 안전하지 않은 방식으로 "__autoload()" 함수를 호출함으로써 공격자가 임의의 서버측 코드를 실행하여 웹 애플리케이션을 장악할 수 있는 취약점

◎ 조치방안

* 취약버전 : PHP 5.3.7 및 5.3.8 이상 패치 적용

[4-3] 쉘 해석기 스크립트 임의 명령 실행

◎ 개요

쉘 해석기는 Unix 기반의 시스템에서 사용하는 쉘 명령어를 해석해주는 도구로, 외부에서 쉘을 실행할 수 있는 디렉터리에 접근한 경우, 원격에서 임의의 명령 수행을 통해 웹서버를 손상시킬 수 있는 취약점

◎ 조치방안

* 웹서버 디렉터리에서 모든 쉘 인터프리터(shell interpreter) 제거
  · 소스 내 Runtime.exec, WSCRIPT.SHELL, shell_exec 함수 제거

[4-4] 매개변수 시스템 호출 코드 인젝션

◎ 개요

웹서버에 요청한 URL의 변수 값에 대하여 정상적인 입력 값 외에 추가적으로 시스템 명령어(e.g. ls, netstat, dump 등)를 삽입하여 요청하는 경우 웹서버가 이 요청을 일반 변수 값으로 인식하여 시스템 명령어가 실행되는 취약점으로 공격자가 외부에서 악의적인 명령어 실행 가능

◎ 조치방안

1 정보출력, 명령어 실행 등 취약한 함수 사용 자제

구분	취약한 함수	
Java(Servlet, JSP)	System.*(특히 System.Runtime)	
Perl	open()	system()
	sysopen()	glob()
PHP	exec()	require()
	system()	include()
	passthru()	eval()
	popen()	preg_replace()

2 부득이 사용해야 하는 경우 입력 값 검증

* 파이프(|), 앰퍼샌드(&), 세미콜론(;) 등 시스템 상에서 멀티라인을 지원하는 특수 문자에 대한 검증

3 미리 정의된 인자 값 배열 생성 후 입력 값에 따라 선택할 수 있도록 정의

안전한 코드 예제

```
……
public void f() throws IOException {
Properties props = new Properties();
String fileName = “file_list”;
FileInputStream in = new FileInputStream(fileName);
props.load(in);
String version[] = {“1.0”, “1.01”, “1.11” , “1.4”};
int versionSelection = Integer.parseInt(props.getProperty(“version”));
String cmd = new String(“cmd.exe   /K ₩”rmanDB.bat ₩””);
String vs = “”;
```

```
// 외부 입력 값에 따라 지정된 목록에서 값을 선택
if (versionSelection == 0)
vs = version[0];
else if (versionSelection == 1)
vs = version[1];
else if (versionSelection == 2)
vs = version[2];
else if (versionSelection == 3)
vs = version[3];
else vs = version[3];
Runtime.getRuntime().exec(cmd + “c:₩₩prog_cmd₩₩” + vs);
……
 }
```

제4장 결론

최근 들어 인터넷 기술이 고도화되면서, 정보개방의 필요성이 급증하고 있다. 이에 공공정보들이 외부로 오픈된 인터넷을 통해 정보유통을 통한 소통의 장으로 활용되어 사용자들은 언제 어디서나 원하는 정보를 편리하게 확인할 수 있는 환경이 제공되고 있다. 하지만 누구에게나 접근이 가능한 인터넷의 구조적인 취약점은 상당히 많은 위험성을 가지고 있으며, 실제 웹 어플리케이션을 활용한 침해사고가 매년 지속적으로 증가하고 있는 추세이다.

특히 '17년도 1분기에는 사용자와 웹서버 간 통신하는 구간에 취약한 암호화 알고리즘 사용, 인증정보 검증 체계 미흡, 개인정보 평문전송 등 개인정보를 보호하기 위한 보호조치가 이루어지지 않아 발생한 취약점이 주를 이루었다. 해당 취약점은 대부분 낮은 버전의 OS가 탑재된 환경에서 SSL 3.0 이하 버전의 암호화 통신 프로토콜을 지원하였으며, MD5, RC4와 같은 쉽게 복호화가 가능한 알고리즘을 사용하여 발생한 것으로 확인되었다.

또한 웹 서버 설계상의 오류나 파일확장자 검증기능의 부재로 인해 외부 비인가자가 임의의 악성파일을 서버에 주입시킬 수 있는 업로드 취약점도 다수 탐지되었는데, 웹 쉘과 같이 서버 쪽에 직접적인 명령을 내릴 수 있는 스크립트가 업로드 될 경우 웹 어플리케이션과 서버 모두 장악될 수 있는 치명적인 위험성이 존재하기 때문에 특별한 주의가 필요하다.

안전한 웹 환경을 구축하고자 한다면 홈페이지 구축단계에서부터의 철저한 사전예방조치가 필요하며, 지속적인 취약점 점검 및 보안조치 수행을 통해 보다 안전한 웹 어플리케이션 환경을 마련하여야 할 것이다.

참고 자료

[1] Secure coding, http://cwe.mitre.org

[2] JAVA, http://wikisecurity.net/guide:java

[3] MS, https://technet.microsoft.com/library/security/ms13-078

[4] MS, http://insecure.org/sploits/Microsoft.frontpage.insecurities.html

[5] REDHAT, https://securityblog.redhat.com/2014/10/15/poodle-a-ssl3-vulnerability-cve-2014-3566/

[6] APACHE, https://tomcat.apache.org/tomcat-6.0-doc/config/http.html#SSL_Support

[7] https://access.redhat.com/ko/node/1258903

[8] https://weakdh.org/

[9] http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2015-0204