WEB VULNERABILITIES EXPLAINED 번역 문서

www.boanproject.com

번역 : 임효영
편집 : 조정원

원본 : http://resources.infosecinstitute.com/web-vulnerabilities-explained/

내용

1. 개요

최근 웹페이지 취약점 문제가 꽤 자주 대두되고 있다. 그 내용은 SQL 인젝션, XSS 취약점, CSRF 등등에 걸쳐 다양하다. 본문에서는 워드프레스를 포함한 웹페이지에서 찾을 수 있는 가장 일반적인 취약점에 대한 기본적인 예시를 제공하고자한다. 다음에서 자세하게 알아보자.

2. 상세 내용

2.1. 서비스 거부 공격

서비스 거부 공격은 여러가지 원인으로 발생한다. 하지만 공격자가 특정 웹 사이트에 악성봇을 사용하여 도스공격을 한다고 설명하기 보다는 오히려 코드를 작성할 때 도스 취약점이 발생할 수 있는지에 대해 설명한다.

일반적으로 우리는 웹 응용 프로그램에서 도스공격이 가능한 시나리오를 만드는 논리적인 실수를 한다. 그러한 시나리오를 짧은 PHP 코드에서 살펴보자. 이 코드는 PHP 샘플 코드로서, 서비스 거부 공격을 유발시키는 논리적인 실수에 대한 내용을 담고 있다.

코드를 보면 먼저 파일 파라메터의 유무 체크를 한다. 있다면 파일 파라메터에 저장된 값을 읽어오고, 없다면 파일 파라메터가 존재하지 않는다는 에러메시지를 표시하면서 프로그램이 닫히게 되어 있다. 따라서 프로그램의 실행을 지속하기 위해서는 파일 파라메터를 제공해야 한다. 그 후에 파일 파라메터에 저장된 값을 읽어오고 파일이 존재 여부를 체크한다. 그렇게 안되면 다시 에러메시지와 함께 응용 프로그램을 중지한다. 하지만 만약 파일이 존재한다면 그것을 현재 실행하고 있는 프로그램에 포함시킨다. 위의 코드를 보면, 해당 코드가 서비스 거부 공격의 결과를 초래할 수 있는 증거가 명백하다.

그럼 위 코드를 index.php 라고 저장하고 파일 파라메터에서 testing.php 의 값을 공급한다고 해보자. 이러한 상황에서는 testing.php 이 존재하고 일부 작업을 수행하는 만큼 모든 작업이 잘 진행된다. 하지만 만약 우리가 index.php 를 파일 파라메터의 값으로 지정한다면 어떻게 될까? 그렇게 한다면 index.php 파일은 현재 실행에 자기자신을 불러오게 되며 결국 서비스 거부로 이어진다. 기본적으로, 운영체제는 너무 많은 메모리를 각 프로그램에 할당하게 되고 프로그램이 더 많은 메모리를 필요로 하면 운영체제가 해당 프로그램을 강제종료한다. 이러한 경우에 일어난다. index.php 파일이 과다 허용된 메모리를 할당한다면 운영체제는 프로그램을 강제 종료한다.

아래는 도스공격을 강제로 프로그램에 보내는 명령어이다.

```
GET /index.php?file=index.php HTTP/1.0
```

2.2. SQL 인젝션(SQL INJECTION)

SQL 인젝션 취약점은 10 년도 전에 알려지긴 했지만 여전히 일반적인 공격방식이다. SQL 인젝션 취약점은 프로그램이 SQL 쿼리나 특수 문자등을 포함하는 입력 값을 따로 캡슐화하지도 않고 따로 검사하지도 않은 상태로 받아들이는 경우 발생한다. 다음은 그러한 프로그램의 예이다.

입력된 ID 와 패스워드가 이에 상응하는 값이 DB 에 존재하는지에 대한 검색이 가능하다. DB 가 해당 ID 와 패스워드를 갖고 있다면, 그 값을 $user 와 $pass 변수로부터 읽어 낼 수 있다. 그리고 ID admin 과 패스워드 admin 을 가지고 로컬 호스트의 SQL 데이터베이스로 접속 할 수 있다.

이후 특수문자가 포함되어 있는지 확인하지 않고 입력된 값으로 구성한 쿼리문을 만든다.

```
<html>
<body>

<?php
  $show = 1;
  if(!empty($_GET['username']) and !empty($_GET['password'])) {
    $user = $_GET['username'];
    $pass = $_GET['password'];

    mysql_connect("localhost", "admin", "admin");
    mysql_select_db("db");
    $result = mysql_query("SELECT * FROM users WHERE username='".$user."' \
      AND password='".$pass."';");
    $row = mysql_fetch_row($result);
    if($row) {
      echo "Welcome user: ".$user;
      $show = 0;
    }
  }
?>

<?php if($show) { ?>
<form action="#">
  Username: <input type="text" name="username" /><br />
  Password : <input type="password" name="password" /><br />
</form>
</body>
</html>
<?php } ?>
```

위 코드는 입력변수에 ID 와 패스워드를 캡슐화하지 않아서 SQL 인젝션에 취약하다는 것을 알아야만 한다. ID 와 패스워크 입력을 ' OR 1=1–'로 했다고 상상해 보라. 그 후에 완성된 SQL 쿼리문은 다음과 같다.

```
SELECT * FROM users WHERE username='' OR 1=1--'' AND password='' OR 1=1--''
```

이 효과적인 쿼리문은 DB 사용자를 모두 선택한다. OR 의 문법을 이용하여 취약한 프로그램에 직접적으로 쿼리명령을 내려 침입한 것이다. 위의 SQL 쿼리는 항상 참이기 때문에 더 이상 진짜 ID 와 패스워드를 입력할 필요가 없다. 대신 프로그램의 논리구조를 깨는 특별한 값을 입력함으로서 로그인 하는 것이다.

2.3. CSRF (CORSS-SITE REQUEST FORGERY)

요청 값 조작 변경(CSRF) 취약점은 타겟 웹사이트에 해당 사이트의 사용자의 이름으로 요청을 심을 수 있을때 나타난다. 그 요청은 타겟 웹 사이트에서 우리가 도용한 사용자의 이름으로 실행된다.

그럼 2 가지 질문을 해보자.

1. 어떻게 하면 사용자에게 요청을 심을 수 있을까?
2. 타겟 웹 페이지에서 어떤 종류의 실행이 가능할까?

첫번째 질문에 대한 답은 간단하다. 최종목적지가 같다는 사실을 통해서 사용자에게 요청을 다양한 방법으로 보낼수 있다.

: 사용자의 브라우저는 타겟 웹사이트에 한 방법이나 또는 다른 방법으로 보내야한다. 우리는 다음방법을 사용하여 사용자에게 요청을 심을 수 있다.

- 타겟 웹 사이트가 취약해서 일시적으로 몇몇 코드를 삽입할수 있는 경우, 사용자가 반드시 클릭할 만한 URI 를 만드는 작업이 필요하다. 그 URI 를 클릭함으로서 최초 요청이 보내진다. 두번째 요청은 해당 사이트의 취약점으로 하여근 공격자가 의도한 행동을 요청하게 만든다.

- 타겟 웹 사이트가 취약해서 사이트에 영구적으로 코드를 삽입할 수 있는 경우에는 간단하게 요청을 웹페이지의 소스코드에 삽입할 수 있다. 사용자가 특정 웹페이지를 방문할 때 설정된 요청이 해당 사용자의 이름으로 실행된다. 이것은 사회공학을 필요로 하지도 않는다. 왜냐하면 모든 사용자가 이 취약한 웹 페이지에 반드시 접근 하기 때문이다.

- 다음으로는 공격자가 완벽하게 제어가능한 웹 페이지를 구성한다. 웹 페이지 단독의 소스코드에서 악성코드 요청을 보낼수 있도록 하는 것이다. 하지만 최종적으로 사용자는 그 웹페이지를 방문한다. 이것이 공격자가 사용자에게 공격자의 웹페이지 링크를 보내야만 하는 이유이다. 사용자가 웹 페이지를 방문하면 공격자의 요청은 웹페이지에 단독으로 첨부되어 사용자의 이름으로 실행된다.

이제 사용자의 브라우저에 공격자의 요청을 심을 수 있는 다양한 방법이 있다는 것을 알았다. 하지만 이것은 50%일 뿐이다. 우리는 웹페이지에 설치 가능한 요청의 종류에 대해 더

알아보고자 한다. 요청된 작업은 우리가 원하는 어떤 것이라도 할 수 있지만 타겟 웹 페이지는 반드시 그 행동과 실행을 그에 따라 지원해야 한다.

그럼 다음 웹페이지의 프로그램에 대해 알아보자

```
<html>
<body>
  <img src="http://www.anything.com/index.php?id=1000&action=up"/>
</body>
</html>
```

사용자가 이 웹 페이지를 방문하면,"http://www.anything.com" 이라는 웹페이지에, id=1000 이며 action=up 라는 파라메터 값을 가지고 index.php 자원을 요청할 것이다. 하지만, 만약 웹 페이지가 index.php 자원이 없는 경우 요청은 실패한다. index.php 가 존재하더라도 id 와 action 의 파라메터 값을 사용하지 않고 있으면 요청은 실패한다. 이것은 우리가 해당 웹페이지에 있는 파일 뿐 아니라 웹 페이지가 작업수행에 필요한 매개 변수를 알고 있어야 한다는 것을 의미한다.

이러한 방법으로 우리는 다른 후보를 대신하여 첫번째 후보자에게 투표하는 요청을 만들어 설문의 결과를 조작 할 수 있다. 그러면 우리는 우리의 페이지로 사용자를 끌어들이고 설문으로의 투표 요청을 보낼수 있다.

하지만 이것은 단지 빙산의 일각일 뿐이다. 만약 어떤 데이타베이스에 다른 관리자를 추가하거나 혹은 모든 사용자 이름을 삭제하거나 심지어 사용자의 메일박스에 있는 주소에 서명된 메일을 보내는 요청을 보내는 것을 상상해봐라.

이제 다른 종류의 유명하고 일반적인 웹 취약점을 둘러보자

2.4. XSS(CROSS-SITE SCRIPTING)

이 공격은 위 설명한 세가지와 마찬가지로 잘 알려진 공격방식이다. 크로스 사이트 스크립팅 공격은 취약한 웹페이지에 임의의 코드를 삽입하여 해당 페이지의 사용자이름, 패스워드, 쿠키정보와 같은 민감한 정보를 빼내는 공격방식이다. XSS 공격을 통해 "모든 스크립트 언어는 사용자의 웹 브라우저에서 실행된다"는 동일 출처 정책을 우회할 수 있다. 이러한 언어의 예가 자바스크립트이다. 동일 출처 정책은 원본의 코드를 가진 웹페이지에만 클라이언트 코드를 웹 브라우저에 실행시키도록 해 준다.

여기에는 세가지 유형의 XSS 공격이 있다.

2.4.1. REFLECTED XSS (반사 XSS)

웹 페이지에서 /"와 같은 특수문자를 금지하는 정책 없이, 사용자의 입력과 출력을 받아들이는 웹 페이지는 취약할 수 밖에 없다. 이 경우 공격자는 자바 스크립트를 포함한 악의적인 의도의 URI 를 사용자에게 보내 해당 링크로 들어가게 한다. 사용자가 링크를 클릭하면, 웹 페이지에 포함된 자바스크립트가 사용자의 브라우저에서 실행된다. 자바스크립트는 사용자의 쿠키정보를 빼내어 공격자에게 보낸다. 이러한 반사 XSS 취약점을 갖고 있는 프로그램의 샘플은 다음과 같다.

```
<html>
<body>

<?php
  if(isset($_GET['p'])) {
    print "V parametru p se nahaja vrednost: " . $_GET['p'];
  }
  else {
    print "Niste nastavili parametra p.";
  }
?>
</body>
</html>
```

먼저 특수문자를 걸러내는 필터링 없이 변수 P 의 값을 설정하고 응답이 나타내는지를 체크한다. 그럼 P 변수의 값속에 자바스크립트 코드를 넣어 사용자가 URI 를 클릭하면 실행되게 지정해 놓을 수 있다. (물론 악성 자바스크립트 코드이다)

다음은 사용자의 쿠키를 빼내오는 자바스크립트를 포함한 URI 의 샘플이다 .

```
GET /index.php?p=<script>alert(document.cookie)</script> HTTP/1.1
```

2.4.2. STORED XSS(저장 XSS)

저장된 XSS 는 악성코드를 취약한 웹페이지에 영구적으로 심을 수 있을 경우에 나타낸다. 따라서 사용자가 해당 취약한 웹페이지를 방문할때마다 악성코드가 실행된다. 악성코드는 웹페이지 자체에 심어져 있기 때문에 공격자는 사용자에게 이메일등을 보내서 링크나 다른것을 클릭하게 할 필요가 없는 편리함이 있다. 이러한 웹 페이지는 반드시 사용자가 입력하는 값을 갖고 있는 특정 종류의 백앤드(후위) 데이터베이스를 사용해야 한다. 사용자가 그 웹페이지를 방문하면 이 값들은 데이터베이스로부터 나와서 웹 페이지에 진열되고, 악성코드를 실행시키게 된다.

2.4.3. DOM-BASED XSS(DOM 기반 XSS)

DOM 기반의 XSS 공격은 공격자가 악성 URI 를 사용자에게 보내어, 사용자가 해당 URI 를 클릭하면 발생한다. 하지만 이것은 앞서 설명한 반영 XSS 공격과는 다르다. 왜냐하면 이

웹사이트는 유효한 비 악성 응답(이 웹사이트는 반영 XSS 공격에 취약하지 않기 때문에)을 결과값으로 보내기 때문이다. 하지만 여기서는 웹사이트가 URI 주소로부터 나오는 값을 사용한 자바스크립트 코드를 쓰기 때문에 공격이 발생된다. DOM 기반의 XSS 의 샘플코드는 다음과 같다.

```
<html>
<body>

<script type="text/javascript">
  p = document.location.href.substring(document.location.href.indexOf("p=")+2);
  document.write("V parametru p se nahaja vrednost: " + p);
</script>

</body>
</html>
```

소스코드를 통해 요청에 대한 응답으로 해당 자바스크립트를 다시 불러오는 것을 알 수 있다. 그 후 자바 스크립트는 사용했던 URI 주소로부터 매개변수 P 의 값을 먼저 읽어들인다. 그 후 매개변수 P 의 값을 표시한다. 이 때문에 자바스크립트가 실제로 악의적인 자바스크립트가 될수 있는 매개변수 P 에 저장된 값을 참조한다는 것이다. 그럼, 다음 요청을 실행한다고 가정해 보자.

```
/index.php?p=<script>alert(document.cookie)</script>
```

응답으로 그 자바스크립트가 실행될 때, **<script>alert(document.cookie)</script>**를 처리과정에 포함하는 매개변수 P 의 값을 읽을 것이다. 따라서 매개변수 P 의 악성코드는 반드시 실행된다. 해당 웹 사이트가 반영이나 저장 XSS 공격에 취약하지 않음에도 불구하고 말이다.

2.5. 버퍼 오버 플로우 (BUFFER OVERFLOW)

때때로 우리는 응용프로그램의 한 부분으로 사용되는 독특한 실행 프로그램을 볼 수 있다. 하지만 그 실행 프로그램이 웹 응용프로그램의 일부로서 사용될 지라도, 버퍼 오버플로우의 취약성은 여전히 존재한다. 웹 응용프로그램이 사용자가 입력한 매개변수와 함께 실행프로그램을 여는 시스템 함수를 요청하는 것을 상상해 보자.

다음 버퍼오버플로우 취약성을 지닌 C 프로그램의 예제를 보자.

```
void copy(char **argv) {
  char array[20];
  strcpy(array, argv[1]);
  printf("%s\n", array);
}

main(int argc, char **argv) {
  copy(argv);
}
```

이 프로그램은 입력 인수를 허용하지만 입력되는 인수의 길이를 확인하지 않는다. 이러한 입력 인수를 허용할 경우 **strcpy 함수** 호출과 함께 로컬 버퍼에 인수를 복사하는 **copy 함수**를 보낼 수

있게 된다. 하지만 로컬 배열에는 겨우 20 바이트 가 예정되어 있기 때문에, 공격자가 20 자를 넘는 인수를 복사하면 버퍼오버플로우가 발생한다. 이것은 프로그램을 중단하게 할 뿐 아니라, 악의적으로 제작된 입력변수라면 타겟 시스템에서 임의의 코드를 실행시킬수 있다.

2.6. 포멧 스트링 (FORMAT STRING)

형식 문자열은 **printf, fprintf, sprintf, snprintf, vprintf, vprintf, vsprintf, vsnprintf** 같은 함수 요청을 사용자가 제어할 수 있는 특별한 취약점이다. 이 공격을 이해하기 위해 아래 예제를 보자. 예제 코드는 C 언어로 쓰여 있으며 포멧 스트링 취약점을 포함하고 있다.

```
#include <stdio.h>

int main(int argc, char **argv) {
  printf(argv[1]);
  return 0;
}
```

위 프로그램에서 **printf 함수**에 직접 전달되는 입력 인수를 허용하고 있는 것을 알 수 있다. 기본적으로 프로그램은 화면에 입력변수를 반복하여 출력하지만 만약 **printf 함수**가 다르게 대응하는 몇몇 특수문자(%n, %x 이나 %s 와 같은)를 입력하면 다르게 출력될 것이다. 만약 우리가 특수문자 %x 를 입력하면 출력함수는 4 바이트의 스택을 출력하고 만약 2 개의 %x 를 입력하면, 8 바이트의 스택을 출력할 것이다. 필수 매개변수 없이 출력함수를 불렀기 때문에 이러한 문제가 생기는 것이다. 하지만 함수는 이 변수의 존재여부와 상관없이 스택(누락된 매개변수가 있어야 하는 자리)으로부터 다음 값을 가져온다.

%x 를 통해 공격자는 스택에 들어있는 임의의 값을 읽어 들일 수 있는 반면, %n 을 통해 임의의 값을 메모리의 임의의 주소에 쓸수도 있다. 이러한 %x 와%n 을 통해서 프로그램의 실행과 임의의 코드의 실행이 가능한 권한을 얻게 된다.

2.7. 디렉터리 검색 공격 (DIRECTORY TRAVERSAL)

작업 디렉토리 접근 취약점은 응용프로그램이 사용자가 읽을 수 있도록 허용되었지만 파일의 적절히 인증이 실패한 파일을 읽어들인 다는데에서 시작한다. 이 취약점은 보통 응용프로그램이 사용자가 읽어오려 하는 파일을 체크하지 않기 때문에 발생한다. 근본적인 원인은 응용프로그램이 사용자가 "../"이나 "..₩"의 특수문자의 사용으로 현재 디렉토리에서 상위 디렉토리로 이동하는 것을 검사하지 않기 때문이다. 따라서 사용자는 허용 디렉터리에서 파일을 읽을 수 없을 뿐 아니라 응용프로그램이 접근해야 하는 다른 모든 디렉토리에 접근할수 없게 된다.

다름 예제는 PHP 로 작성된 취약한 응용프로그램이다.

```
<?php
if(empty($_GET['file']))
```

```
   die('You didn\'t enter the name of the file.');

$file = getcwd().'/'.$_GET['file'];
if(!file_exists($file))
   die('The filename doesn\'t exist.');

readfile($file);
?>
```

위의 코드에서 우리는 먼저 매개변수 **file** 이 존재하는지 그리고 값을 갖고 있는지 검사한다. 현재 디렉토리를 얻어내고 매개변수 P 의 값을 추가시키는 **getcwd** 함수를 사용하여 읽어내고자 하는 파일에게 도달하는 전체 경로를 구축한다. 마지막으로 구축된 경로를 사용해 파일을 읽어 들이고 사용자에게 표시한다.

문제는 매개변수 P 속에 들어있는 침입의 우려가 있는 악성 문자를 검사하지 않는 곳에서 발생한다. 이것은 공격자에게 "../"나 "..₩"등의 특수 문자를 사용해 디렉토리 트리를 탐색하는 것을 허용합니다. 응용프로그램이 해당 파일에 접근할 수 없는 경우라도 시스템의 /etc/password 파일이 유출될 수 있다. 이 요청은 다름과 같은 모습일 것이다.

```
GET /index.php?file=../../../../etc/passwd HTTP/1.0
```

2.8. 파일 삽입 공격 (FILE INCLUSION)

파일 삽입 공격은 디렉토리 검색 공격과 매우 비슷하다. 유일한 차이점은 디렉토리 검색 공격에서는 읽기 허용되어 있지 않은 파일만을 읽을 수 있지만 파일 삽입 공격에서는 현재 웹 페이지에서 실행되고 있는 파일을 포함하여 읽을 수 있다는 것이다. 따라서 우리가 실행할 권한이 없는 파일을 실행한다는 것이다.

다음 두가지 종류의 파일 삽입 공격을 살펴보자

- **로컬 파일 삽입 공격 (LFI : Local File Inclusion)**

여기에서는 현재 실행중인 로컬 파일을 포함한다. 로컬 파일에 의해 해당 파일이 현재 서버의 시스템 위에 존재하는 파일임을 알 수 있다.응용 프로그램이 사용자가 입력한 데이터를 캡슐화 하지 않기 때문에 로컬 파일 삽입 공격이 가능하게 된다는 것이다.

- **원격 파일 삽입 공격 (RFI : Remote File Inclusion)**

여기에서는 현재 실행되고 있는 원격 파일을 포함한다. 응용프로그램이 서버 파일시스템으로의 파일을 업로드하는 옵션이 있을 경우 여기에 해당된다. 이러한 경우, 악성파일을 사용자로부터 서버에 업로드 할 수 있고 그 악성 파일을 실행할 수 있다. 이 공격을 통해 악성 웹 쉘을 취약한 응용프로그램에 업로드하고 웹서버로서의 전체적인 제어가 가능하게 된다.

이러한 취약한 코드의 예는 다음 소스코드의 출력에서 볼수 있다.

```php
<?php
if(empty($_GET['file']))
  die('You didn\'t enter the name of the file.');

$file = getcwd().'/'.$_GET['file'];
if(!file_exists($file))
  die('The filename doesn\'t exist.');

include($file);
?>
```

위의 코드에에서는 우선 매개변수 파일이 존재하고 값을 포함하는지를 체크한다. 그 후 읽어내고자 하는 파일을 위해 현재 디렉토리를 선택하고 거기에 매개변수 P 의 값을 첨부하는 **getcwd** 함수를 사용하여 전체 경로를 구축한다. 마지막으로 구축된 경로의 파일을 웹사이트에 현재 실행시키는 것으로 포함한다.

2.9. 명령어 삽입 (COMMAND INJECTION)

명령 삽입 취약점은 사용자가 입력한 값이 서버에서 실행될 명령에 영향을 줄 때 발생한다. PHP 로 작성된 해당 취약점의 예제는 다음과 같다.

```php
<html>
<body>
<?php
  if(empty($_GET['user']))
    die('You didn\'t enter the name of the user.');
  $user = $_GET['user'];
  system("id $user");
?>
</body>
</html>
```

위 코드에서는 우리가 먼저 만약 매개변수 사용자가 존재하고, 그것이 동작하는 지를 체크한 흐 매개변수의 값을 로컬 변수에 읽어들인다. 그 후에 시스템에서 명령 "ID user" 를 실행한다. 이 취약점은 응용프로그램이 입력되는 값에 특수문자의 유무를 검사하지 않기 때문에 일어난다.

우리가 **"root;ls -l /"**라는 값을 투과 없이 입력변수에 입력하면 두번째 명령을 시스템에서 실행시킬수 있기 때문이다. 그러면 응용 프로그램은 **"id root;ls -l /"** 라는 명령을 실행할 수 있다. 하지만 ";"가 두 개 이상의 명령을 분리하는 역할을 해주기 때문에 응용프로그램은 사실 두개의 명령을 한개씩 실행하고 두개의 결과값을 리턴하게 된다.

2.10. 권한 상승 (PRIVILEGE ESCALATION)

권한 상승 공격은 개발자가 의도하지 않는 권한을 공격자가 가질 수 있는 논리적 오류를 사용한 공격이다. 이 취약점은 종종 허가받지 않은 사용자,허가된 사용자, 관리자 등과 같은 여러 권한을 한 사용자가 담당할 경우 발생한다. 몇몇 사용자의 역할은 원래 갖고 있어야 할 권한 보다 더 많은 권한을 가질 필요는 없다. 예를 들어 관리자는 다른 사용자 계정을 추가 할 수 있는 권한이

있어야 한다. 하지만 다른 사용자는 그런 권한이 있어서는 안된다. 그럼에도 불구하고 일반사용자에게 새사용자 계정을 만드는 권한이 있다면 취약점이 발생한다.

3. 결론

우리는 웹 기반 응용 프로그램을 프로그래밍 할때 조심해야 하는 많은 취약점이 있다는 것을 알았다. 가장 일반적인 취약점의 기본 예제를 보면서 취약점을 막을 수 있는 더 나은 방법에 대해 설명하였다. 웹 개발자가 공부하고 사용할수 있는 많은 응용 프로그램의 일반적인 실수에 대한 참고가 될 수 있기를 바란다.

관련된 컨텐츠

- hacker site wallpaper
- hacking wallpaper
- most common web exploits file name
- php argv parameter injection
- web vulnerabilities explain infosec edu
- web vulnerabilities explained

저자에 관하여

Dejan Lukan 은 InfoSec Institute 의 보안 리서치이며 Slovenia 에서 모의 침투 테스터로 활동하고 있다. 소스코드레벨, 퍼징, 역공학분석과 같이 상용 응용 어플리케이션의 취약점을 도출하는데 관심이 많다. 보안과 관련된 문제점에 대해서 간단하게나마 자신의 스크립트를 제작하기도 하고 새로운 해킹 기법에 대해 연구를 하고 있다. 수많은 관련 코드를 제작할 정도로 프로그래밍 언어에 숙달되어 있다. 또한, 안티 바이러스 우회 기법들, 악성코드 리서치, 리눅스 기반, 윈도우즈, BSD 기반 장비 운영에 강점이 있다.

2. 끝맺음

http://resources.infosecinstitute.com/ 사이트에서는 다양한 해킹 공격 시연 문서 및 방어들이 정기적으로 배포되고 있습니다. 입문자들 대상으로 설명한 문서들이 많아서 연구 목적으로 번역을 시작하였습니다. 앞으로도 좋은 컨텐츠에 대해서는 정기적으로 번역을 해서 배포하도록 하겠습니다.

또한 꾸준히 활동하는 멤버들에게 번역출판의 기회를 드리고 있습니다. 번역에 참여해주신 멤버들에게 감사합니다.