시큐어 코딩 기법

2024-05-22 IT정보공학과 이진규

Secure Coding

소프트웨어 개발 시 안전한 코드를 제작해 취약점을 방지하고

공격으로부터 보호하기 위한 코딩 기법 및 프로세스

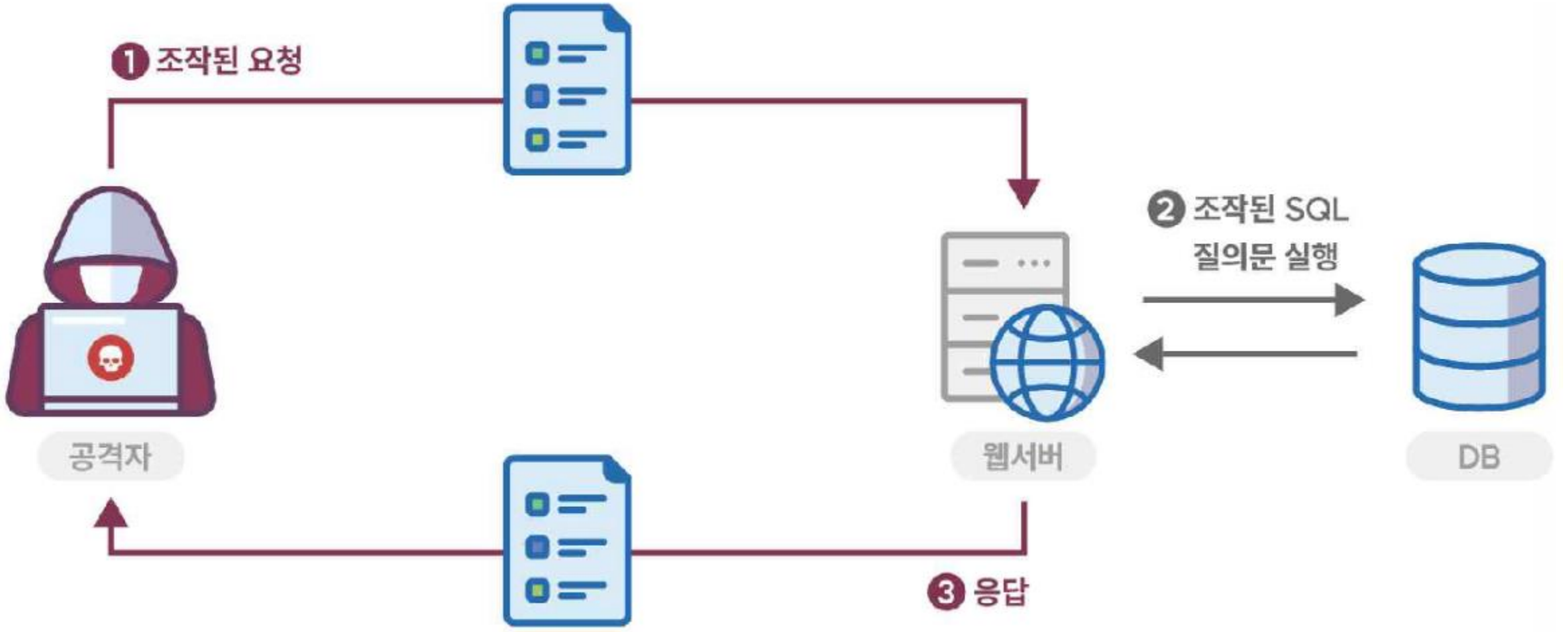
Secure Coding 중요성

침입탐지 시스템, 안티 바이러스 등의 제품을 통해서 보안 수준을 강화

but 단순 제품 도입만으로는 소프트웨어의 보안 취약점을 활용한 공격에 대해서 대응하기 어려움

시큐어 코딩을 통해서 보안 취약점을 사전에 방지

SQL Injection



공격자가 악의적인 요청을 통해 잘못된 SQL 쿼리를 요청, 실행

의도와 다른방식으로 DB 동작

SQL Injection


```python
def update_board(request):
    dbconn = connection

    with dbconn.cursor() as curs:
        # 사용자의 입력값 받음
        name = request.POST.get('name', '')
        content_id = request.POST.get('content_id', '')

        # 입력값을 통해 동적 쿼리문 생성
        sql_query = "update board set name='" + name + "'where contend_id='" + content_id + "'"

        # 외부 입력값을 검증 없이 쿼리로 포함하여 안전하지 않음
        curs.execute(sql_query)
        dbconn.commit()

        return render(request, '/success.html')
```

db 정보 노출 및 손상

sql_query -> 입력값 들어간다 알려줌

입력값에 해당하는 매개변수를 넣고 실행

```python
def update_board(request):
    dbconn = connection

    with dbconn.cursor() as curs:
        # 사용자의 입력값 받음
        name = request.POST.get('name', '')
        content_id = request.POST.get('content_id', '')

        # 입력값 조작으로부터 안전한 인자화된 쿼리를 생성
        sql_query = 'update board set name=%s where content_id=%s'

        # 사용자의 입력값이 인자화된 쿼리로 실행되므로 안전
        curs.execute(sql_query, (name, content_id))
        dbconn.commit()

        return render(request, '/success.html')
```

Path Traversal

파일 경로를 입력받게되는 url 을 사용하면 값을 조작해 악용 가능

Path Traversal


```python
def get_info(request):
    # 외부 입력값으로부터 파일명을 입력 받음
    request_file = request.POST.get('request_file')
    # 파일명, 확장자 파싱
    (filename, file_ext) = os.path.splitext(request_file)
    file_ext = file_ext.lower()

    if file_ext not in ['.txt', '.csv']:
        return render(request, '/error.html', {'error':'can\'t open'})

    # 입력값을 검증 없이 파일 처리에 사용
    with open(request_file) as f:
        data = f.read()

    return render(request, '/success.html', {'data':data})
```

확장자에 대한 확인만 존재

입력값에 대해 검증을 하지 않고 파일 데이터를 읽어와 파일 처리

```python
def get_info(request):
    # 외부 입력값으로부터 파일명을 입력 받음
    request_file = request.POST.get('request_file')
    # 파일명, 확장자 파싱
    (filename, file_ext) = os.path.splitext(request_file)
    file_ext = file_ext.lower()

    if file_ext not in ['.txt', '.csv']:
        return render(request, '/error.html', {'error':'can\'t open'})

    # 파일 명에서 경로 조작 문자열을 필터링
    filename = filename.replace('.', '')
    filename = filename.replace('/', '')
    filename = filename.replace('\\', '')
    try:
        with open(filename + file_ext) as f:
            data = f.read()
    except:
        return render(
        request, "/error.html", {"error": "not exist or can\'t open"}
        )
    return render(request, '/success.html', {'data':data})
```

파일명에 해당하는 부분에 '.' '/' '\\' 등의 문자열 필터링

사용자가 임의로 url 조작하여 악용하는것을 방지

XSS

html 태그 사이에 script를 넣어 악의적인 행위를 함

공격 스크립트가 포함된 내용을 게시글에 등록하게되면
script문을 수행해 민감한 정보등을 게시글을 통해 확인 가능

XSS


```python
def profile_link(request):
    # 외부 입력값을 검증 없이 HTML 태그 생성의 인자로 사용
    profile_url = request.POST.get('profile_url')
    profile_name = request.POST.get('profile_name')
    # 입력값에 </a> <script>'공격 스크립트'</script> <a>
    object_link = '<a href="{}">{}</a>'.format(profile_url, profile_name)

    # mark_safe함수는 Django의 XSS escape 정책을 따르지 않는다
    object_link = mark_safe(object_link)

    return render(request, 'my_profile.html',{'object_link':object_link})
```

mark_safe는 텍스트에 대해 escape를 생략

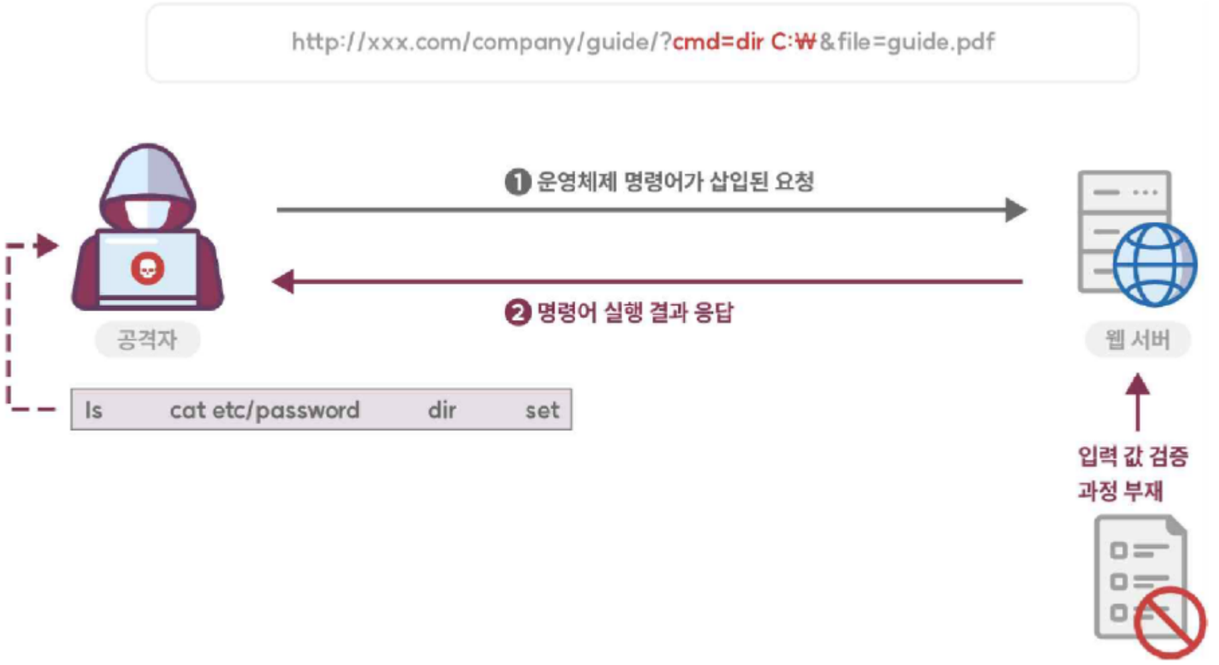
Django는 모든 텍스트를 escape시켜 전달

아무리 포맷에 맞는 html 코드 문자열을 넘겨주어도 텍스트 그 자체로 출력, html 변환 X

```python
def profile_link(request):
    # 외부 입력값을 검증 없이 HTML 태그 생성의 인자로 사용
    profile_url = request.POST.get('profile_url')
    profile_name = request.POST.get('profile_name')
    # 입력값에 </a> <script>'공격 스크립트'</script> <a>
    object_link = '<a href="{}">{}</a>'.format(profile_url, profile_name)

    return render(request, 'my_profile.html',{'object_link':object_link})
```

Django에서 XSS가 발생하지 않도록 설정 되어있음

Command Injection



웹 서버에 공격자가 리눅스 명령어를 입력해 실행 시킴

여러가지 명령어들을 서버에서 실행시킬 수 있도록 만듦

Command Injection


```python
def execute_command(request):
    app_name_string = request.POST.get('app_name','')

    # 입력 파라미터를 제한하지 않아 입력값으로 전달된
    # 모든 프로그램이 실행될 수 있음
    os.system(app_name_string)

    return render(request, '/success.html')
```

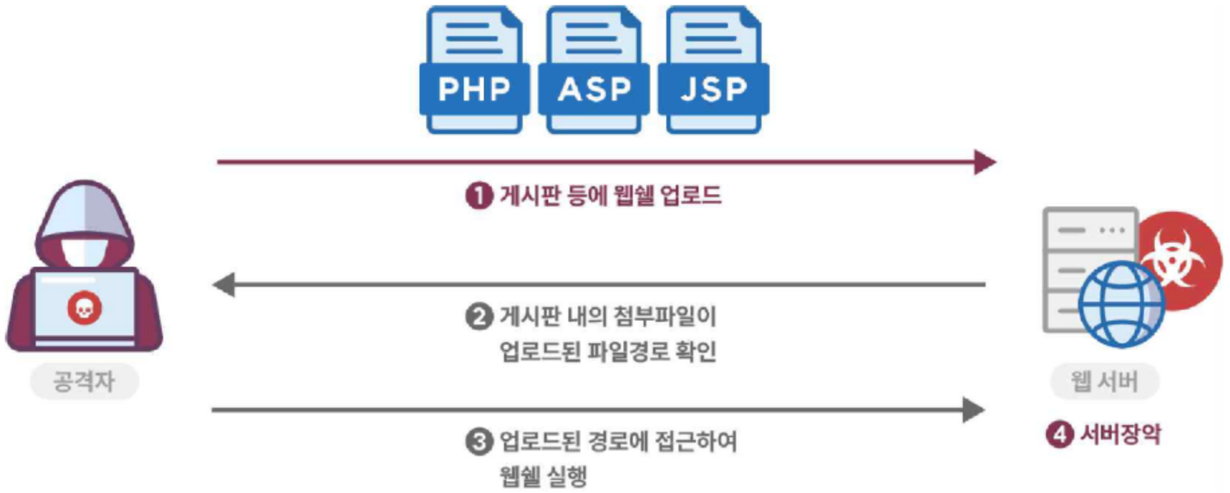
os.system을 통해 실제로 리눅스 명령어를 실행할 수 있음

```python
ALLOW_PROGRAM = ['notepad', 'calc']

def execute_command(request):
    app_name_string = request.POST.get('app_name','')

    # 입력받은 파라미터가 허용된 시스템 명령어 목록에 포함되는지 검사
    if app_name_string not in ALLOW_PROGRAM:
        return render(request, '/error.html', {'error':'not allow'})

    os.system(app_name_string)

    return render(request, '/success.html')
```

입력받은 값이 리스트에 있는지 확인

개발자가 의도한 명령어만 실행할 수 있도록 리스트 추가

File Upload



게시판에 등에 악의적인 웹쉘을 업로드하여 웹서버에 공격

File Upload


```python
def file_upload(request):
    if request.FILES['upload_file']:
        # 사용자가 업로드하는 파일을 검증 없이 저장
        upload_file = request.FILES['upload_file']

        fs = FileSystemStorage(location='media/screenshot', base_url='media/screenshot')
        # 업로드 하는 파일에 대한 여러 요소 등을 검증하지 않음
        filename = fs.save(upload_file.name, upload_file)
        return render(request, '/success.html', {'filename':filename})

    return render(request, '/error.html', {'error':'fail to upload'})
```

파일 업로드 시 파일에 대한 어떠한 검증도 없음

공격자는 웹쉘 과 같은 악의적인 파일 업로드 가능

```python
def file_upload(request):
    # 파일 개수 제한
    if len(request.FILES) == 0 or len(request.FILES) > FILE_COUNT_LIMIT:
        return render(request, '/error.html', {'error': 'fail'})

    filename_list = []
    for filename, upload_file in request.FILES.items():
        # 파일 타입 확인
        if upload_file.content_type != 'image/jpeg':
            return render(request, '/error.html', {'error': 'type error'})

        # 파일 크기 제한
        if upload_file.size > FILE_SIZE_LIMIT:
            return render(request, '/error.html', {'error': 'size error'})
        # 파일 확장자 검사
        file_name, file_ext = os.path.splitext(upload_file.name)
        if file_ext.lower() not in WHITE_LIST_EXT:
            return render(request, '/error.html', {'error': 'ext error'})

        fs = FileSystemStorage(location='media/screenshot', base_url = 'media/screenshot')
        for upload_file in request.FILES.values():
            filename = fs.save(upload_file.name, upload_file)
        filename_list.append(filename)
    return render(request, "/success.html", {"filename_list": filename_list})
```

파일 특성에 대한 제한을 둠

개발자가 의도한 파일들만 업로드 가능

하드코딩

사용자 id, 경로 등 민감한 값들을 코드에 노출하는 경우 위험

하드코딩


```python
def query_execute(query):
    # user, passwd가 소스코드에 평문으로 하드코딩 됨
    dbconn = pymysql.connect(
    host='127.0.0.1',
    port='1234',
    user='root',
    passwd='1234',
    db='mydb',
    charset='utf8',
    )
    curs = dbconn.cursor()
    curs.execute(query)
    dbconn.commit()
    dbconn.close()
```

사용자 정보들을 문자열로 직접 입력

공격자가 코드에 접근하면 해당 정보들을 확인할 수 있음

```python
def query_execute(query, config_path):
    with open(config_path, 'r') as config:
        # 설정 파일에서 user, passwd를 가져와 사용
        dbconf = json.load(fp=config)

        # 암호화되어 있는 블록 암호화 키를 복호화 해서 가져오는 사용자 정의 함수
        blockKey = get_decrypt_key(dbconf['blockKey'])

        # 설정 파일에 암호화되어 있는 값을 가져와 복호화한 후에 사용
        dbUser = decrypt(blockKey, dbconf['user'])
        dbPasswd = decrypt(blockKey, dbconf['passwd'])

    dbconn = pymysql.connect(
    host=dbconf['host'],
    port=dbconf['port'],
    user=dbUser,
    passwd=dbPasswd,
    db=dbconf['db_name'],
    charset='utf8',
    )
    curs = dbconn.cursor()
    curs.execute(query)
    dbconn.commit()
    dbconn.close()
```

민감한 사용자 정보들을 따로 파일에 암호화하여 저장 후 값들을 가져와야 하는 경우에 복호화하여 사용

반복된 인증시도 제한

공격자가 패스워드를 맞추기 위해 모든 값들을 넣어봄

반복된 인증시도 제한


```python
def login_limit(request, uid):
    with connection.cursor() as cursor:
        cursor.execute("SELECT userLocked, loginLimit FROM test_user WHERE userid = %s", [uid])
        row = cursor.fetchone()

    if row['userLocked'] == 1:
        messages.error(request, "Call manager")
        return redirect('index')

    # 6회 이상 틀릴 시 계정 잠김
    login_limit_left = 5 - row['loginLimit']
    if login_limit_left <= 0:
        with connection.cursor() as cursor:
            cursor.execute("UPDATE test_user SET userLocked = 1 WHERE userid = %s", [uid])

        messages.error(request, "Call manager")
        return redirect('index')
    else:
        messages.error(request, f"Login failed {login_limit_left} left.")
        return redirect('index')
```

n회 이상 틀릴 시 계정 잠김

계정을 풀기 위해서 관리자의 조치 필요

loginLimit	userLocked
0	0
0	0
0	0
0	0
0	1
0	0
NULL	NULL

데이터베이스 활용

감사합니다