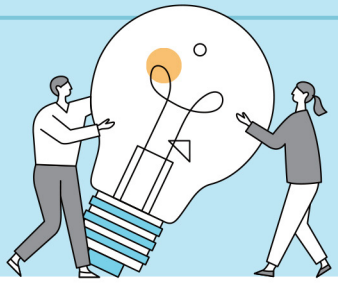
Python 시큐어코딩 가이드

2022.02

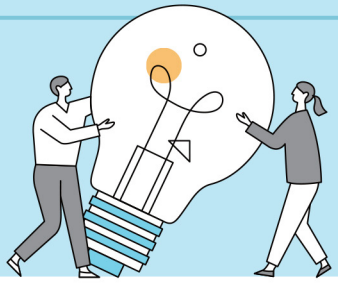
과학기술정보통신부

KISA 한국인터넷진흥원

COTENTS



COTENTS



PART 제3장 부록

1

PART 제1장

개 요

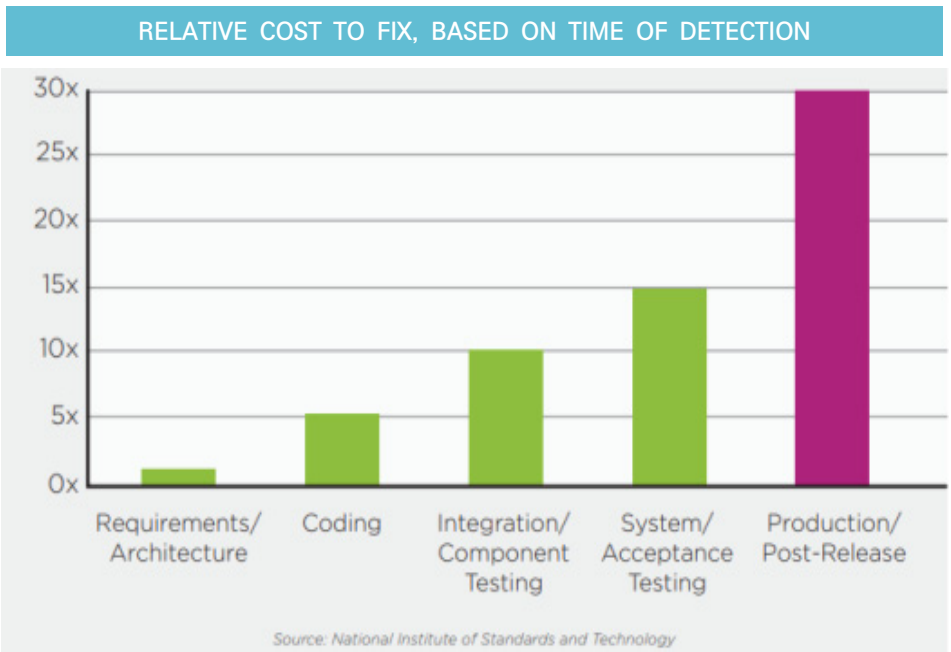
1 개요

제1절 배경

공격자의 초점이 지속적으로 애플리케이션 계층을 향해 이동함에 따라 소프트웨어 자원보호가 중요해졌다. 최근 발생되는 인터넷상 공격시도의 약 75%는 소프트웨어보안취약점을 악용하는 것으로, 특히 외부에 공개되어 불특정다수를 대상으로 사용자 정보를 처리하는 웹 애플리케이션의 취약점으로 인해 중요정보가 유출되는 침해사고가 빈번하게 발생되고 있다. 보안강화를 위해 구축해놓은 침입차단시스템과 같은 보안장비로 응용프로그램 취약점에 대한 공격을 완벽히 방어하는 것은 불가능하다.

소프트웨어의 개발보안 미적용으로 제품 출시 때 수정 시 설계 단계보다 30배, 구현 단계 보다 20배가 넘는 수정비용이 필요 한 걸로 조사되었다.

탐지시간을 기준으로 한 상대적 수정비용



* 출처 : VERACODE – Secure Development Survey: Developer's Respond To Application Security Trends (2021)

‘소프트웨어 개발보안’은 소프트웨어 개발과정에서 개발자의 실수, 논리적 오류 등으로 인해 발생될 수 있는 보안 취약점, 보안약점들을 최소화하여 사이버 보안위협에 대응할 수 있는 안전한 소프트웨어를 개발하기 위한 일련의 보안활동을 의미한다. 즉, 소프트웨어 개발 생명주기(SDLC, Software Development Life Cycle)의 각 단계별로 요구되는 보안활동을 수행함으로써 안전한 소프트웨어를 만들 수 있도록 한다.

소프트웨어 개발보안의 중요성을 인식한 미국의 경우 국토안보부(DHS)를 중심으로 시큐어코딩을 포함한 소프트웨어(SW) 개발 전 과정(설계, 구현, 시험 등)에 대한 보안활동 연구를 활발히 진행하고 있으며, 이는 2011년 발표한 “안전한 사이버 미래를 위한 청사진(Blueprint for a Secure Cyber Future)”에 나타나있다.

국내의 경우, 2009년부터 전자정부서비스 개발단계에서 소프트웨어 보안약점을 진단하여 제거하는 소프트웨어 개발보안(시큐어코딩) 관련 연구를 진행하면서, 2012년까지 전자정부지원사업 등을 대상으로 소프트웨어 보안약점 시범진단을 수행하였다. 이러한 소프트웨어 보안약점 제거·조치 성과에 따라 2012년 6월에 행정안전부 ‘정보시스템 구축·운영 지침’이 개정·고시됨으로써 전자정부서비스 개발 시 적용토록 의무화 되었다. 2019년 6월에는 소프트웨어 개발 보안의 법적 근거를 담은 소프트웨어진흥법 개정안이 발의되었고 2020년 12월 10일에 시행됨에 따라 민간분야에서의 소프트웨어 개발보안 영역이 확대되었다.

과학기술정보통신부는 디지털뉴딜의 성공과 정보보호 패러다임 변화에 대응하고 안전하고 신뢰할 수 있는 디지털 안심 국가 실현을 목표로 K-사이버방역 체계를 2021년 2월에 발표하였으며, 민간의 안전한 디지털 전환을 돕고자 개발보안 컨설팅 제공 및 SW안전성 점검을 지원하고 있다.

제2절 가이드 목적 및 구성

빠르게 변화하는 ICT기술 환경에 발맞추어 민간에서 다양한 분야에 걸쳐 사용되는 언어에 대한 보안 가이드의 필요성이 높아졌다. 이에 따라 공공분야에서는 행정안전부 주관 KISA에서 발간한 소프트웨어 개발보안 가이드 및 프로그래밍 언어별(C, Java, Android-Java) 시큐어코딩 가이드 등을 개발하여 보급하였다.

기존 가이드에서 다룬 JAVA, C의 경우 Web, Cloud, Back-End 서비스 그리고 Embedded 등 공공분야에서 사용되는 서비스에 대하여 시큐어코딩 방법을 제시하고 있다.

프로그래밍 언어의 종류가 다양해짐에 따라 민간에서 타 언어에 대한 시큐어코딩 가이드에 대한 수요가 늘어났다. 이에 민간에서 가장 많이 활용되고 있는 언어를 조사하여 그 중 선호도가 가장 높은 Python 언어에 대한 시큐어코딩 가이드를 제작하게 되었다.

Python은 1991년에 발표된 고급언어로 플랫폼에 독립적이며 인터프리터식, 객체지향적, 동적 타이핑(dynamically typing) 대화형 언어이다. 다양한 플랫폼에서 쓸 수 있고, 라이브러리(모듈)가 풍부하여 대학을 비롯한 여러 교육 기관, 연구기관 및 산업계에서 이용이 증가하고 있다. 최근 웹 개발 이외에도 그래픽, 머신러닝 업계에서 선호하는 언어로 최근은 C, JAVA를 제치고 선호도 1위에 오르기도 했다.

TIOBE Programming Community Index

Nov 2021	Nov 2020	Change	Programming Language	Ratings	Change
1	2	^	Python	11.77%	-0.35%
2	1	v	C	10.72%	-5.49%
3	3		Java	10.72%	-0.96%
4	4		C++	8.28%	+0.69%
5	5		C#	6.06%	+1.39%

* 출처 : tiobe - Python Programming Language of the Year (2021.11.)

본 가이드는 Python 소프트웨어 개발 시 고려되어야 하는 보안 위협을 최소화하기 위해 구현 단계에서 검증해야 하는 보안약점들에 대한 설명과 안전한 코딩 기법, 관련 코드예제를 제공하여 안전한 소프트웨어 개발에 도움이 되도록 한다. Python 3.X 버전을 기준으로 작성되었으며, 구현단계 보안약점 제거 기준 항목 중 언어 특성에 따라 일부 항목은 제외되어 있다.

목적

- 안전한 Python 소프트웨어 개발을 위한 시큐어코딩 방법 제시

구성

- **(1장) Python 개발보안 가이드 개발 배경 및 목적**
- **(2장) Python 언어 기반 구현단계 보안약점 제거 기준 설명**
  - 구현단계 보안약점 제거 기준 항목(49개) 중 42개에 대해 소개

유형	주요 내용
입력데이터 검증 및 표현	• SQL 삽입, 코드 삽입, 경로 조작 및 자원 삽입 등 15개 항목 ※ (2개 항목 제외) 정수형 오버플로우, 메모리 버퍼 오버플로우
보안기능	• 적절한 인증 없는 중요 기능 허용, 부적절한 인가 등 14개 항목 ※ (2개 항목 제외) 부적절한 전자서명 확인, 부적절한 인증서 유효성 검증
시간 및 상태	• 경쟁조건, 종료되지 않는 반복문 또는 재귀함수 2개 항목
에러처리	• 오류 메시지 정보노출, 오류상황 대응 부재 등 3개 항목
코드오류	• Null Pointer 역참조, 부적절한 자원 해제 등 3개 항목 ※ (2개 항목 제외) 해제된 자원 사용, 초기화되지 않은 변수 사용
캡슐화	• 잘못된 세션에 의한 데이터 정보노출 등 4개 항목
API 오용	• DNS lookup에 의한 보안결정 1개 항목 ※ (1개 항목 제외) 취약한 API 사용

- **(3장) 구현단계 보안약점 제거 기준 및 용어 설명**

활용

- Python 기반 시큐어코딩을 적용한 소프트웨어 개발 시 참조
- Python 기반 보안약점 및 대응기법 등 전반에 대한 이해

2

PART 제2장

시큐어코딩 가이드

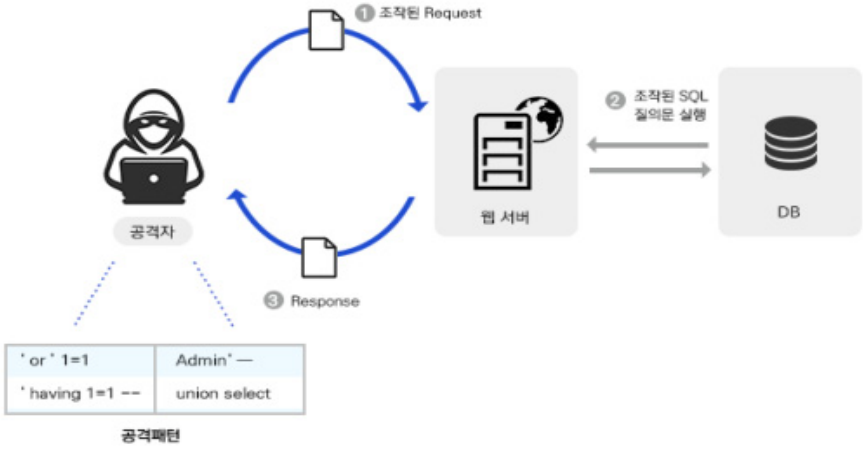
2 시큐어코딩 가이드

제1절 입력데이터 검증 및 표현

프로그램 입력 값에 대한 검증 누락 또는 부적절한 검증, 데이터의 잘못된 형식지정, 일관되지 않은 언어셋 사용 등으로 인해 발생되는 보안약점으로 SQL 삽입, 크로스사이트 스크립트(XSS) 등의 공격을 유발할 수 있다.

1. SQL 삽입

가. 개요



데이터베이스(DB)와 연동된 웹 응용프로그램에서 입력된 데이터에 대한 유효성 검증을 하지 않을 경우, 공격자가 입력 폼 및 URL 입력란에 SQL 문을 삽입하여 DB로부터 정보를 열람하거나 조작할 수 있는 보안취약점을 말한다. 취약한 웹 응용프로그램에서는 사용자로부터 입력된 값을 검증 없이 넘겨받아 동적쿼리(Dynamic Query)를 생성하기 때문에 개발자가 의도하지 않은 쿼리가 실행되어 정보유출에 악용될 수 있다.

Python에서는 데이터베이스에 엑세스에 사용되는 다양한 Python 모듈간의 일관성을 장려하기 위해 DB-API를 정의 하고 있고 각 데이터베이스마다 별도의 DB 모듈을 이용해 데이터베이스에 엑세스하게 된다. DB-API 외에도, 파이썬에서는 Django, SQLAlchemy, Storm등의 ORM을 사용하여 데이터베이스에 엑세스할 수 있다.

Python에서 지원하는 다양한 ORM(Object Relational Mapping)을 이용하여 보다 안전하게 DB를 사용할 수 있지만 일부 복잡한 조건의 쿼리문 생성 어려움, 성능저하 등의 이유로 쿼리의 튜닝이 필요한 경우 직접 원시 SQL 실행이 필요한 경우가 있다. ORM 대신 원시 쿼리를 사용하는 경우 검증되지 않은 외부 입력값으로 인해 SQL 삽입 공격이 발생할 수 있다.

나. 안전한 코딩기법

DB-API를 사용할 때에는 매개변수화된 쿼리(Parameterized query)를 사용하여 외부 입력값을 바인딩해서 사용하면 SQL 삽입 공격으로부터 안전하게 사용할 수 있다.

Python에서 많이 사용되는 ORM프레임워크로는 Django의 querySets, SQLAlchemy, Storm등이 있다. ORM 프레임워크는 내부적으로 사용되는 쿼리 모든 곳에서 매개변수화된 명령문을 사용하므로 SQL 삽입 공격으로부터 보호된다.

ORM 프레임워크의 원시 SQL을 사용할 경우에도 안전하게 사용하려면 외부 입력 값을 매개변수화된 쿼리문의 바인딩 변수로 사용한다.

다. 코드예제

가) DB-API 사용 예제

다음은 MySql, PostgreSQL의 DB-API를 사용하여 입력 값을 받아서 처리 하는 안전하지 않은 코드이다. 이 경우에는 외부 입력값을 8, 9 라인에서 입력 받아 변수 name과 content_id에 할당했다. 12 ~ 15라인에서는 외부에서 입력 받은 name과 content_id값을 검증 없이 쿼리문의 인자 값으로 사용하는데 단순 문자열 결합을 통해 쿼리를 생성하고 있다. 이 경우 content_id 값으로 'a' or 'a' = 'a와 같은 공격 문자열을 입력하면 조건 절이 content_id = 'a' or 'a' = 'a'로 바뀌고, 그 결과 board 테이블 전체 레코드의 name 컬럼이 공격자로부터 입력 받은 name의 값으로 변경된다.

안전하지 않은 코드의 예

```
1:  from django.shortcuts import render
2:
3:  def update_board(request):
4:      ......
5:
6:      with dbconn.cursor() as curs:
7:          # 외부로부터 입력받은 값을 검증 없이 사용할 경우 안전하지 않다.
8:          name = request.POST.get('name', '')
9:          content_id = request.POST.get('content_id', '')
10:
11:         # 사용자의 검증되지 않은 입력으로 부터 동적으로 쿼리문 생성
12:         sql_query = "update board set name='" + name + "' where content_id='" + content_id + "'"
13:
14:         # 외부 입력값이 검증 없이 쿼리로 수행되어 안전하지 않다.
15:         curs.execute(sql_query)
16:         curs.commit()
17:         return render(request, '/success.html')
```

다음은 이를 안전한 코드로 변환한 예제이다. 7, 8라인에서 입력받은 외부 입력값을 그대로 사용하지 않고, 15라인의 execute()메서드의 두 번째 인자 값으로 바인딩 해서 쿼리 문을 실행 하였다. 11라인과 같이 매개변수 바인딩을 통해 execute() 함수를 호출하면 공격자가 쿼리를 변조하는 값을 삽입하더라도 해당 값이 바인딩된 매개변수의 값으로만 사용되지 때문에 안전하다.

안전한 코드의 예

```
1:  from django.shortcuts import render
2:
3:  def update_board(request):
4:      ......
5:
6:      with dbconn.cursor() as curs:
7:          name = request.POST.get('name', '')
8:          content_id = request.POST.get('content_id', '')
9:
10:         # 외부 입력값으로 부터 안전한 매개변수 화된 쿼리를 생성 한다.
11:         sql_query = 'update board set name=%s where content_id=%s'
12:
13:         # 사용자의 입력 값을 매개변수 화된 쿼리에 바인딩 하여 실행되므로
14:         # 안전하다.
15:         curs.execute(sql_query, (name, content_id))
16:         curs.commit()
17:         return render(request, '/success.html')
```

SQLite DB-API 사용에서도 동일하게 정적인 쿼리문을 사전에 생성하고 사용자 입력을 바인딩 하여 안전하게 사용하여야 한다. SQLite에서 매개변수 화된 쿼리(Parameterized Query)를 만들기 위해 “?”를 Placeholder로 사용하거나 “:name” 처럼 Named Placeholder를 사용하는 방법 2가지가 있다.

나) ORM 사용 예제

Django의 querysets는 쿼리 매개변수화를 사용하여 쿼리를 구성하기 때문에 SQL 삽입 공격으로부터 보호된다. 부득이하게 원시 SQL, 또는 사용자 정의 SQL을 사용할 경우에도 외부 입력값을 매개변수화된 쿼리의 바인딩 변수로 사용하여야 한다.

아래는 Django의 원시 SQL을 사용하는 예제이다. Django의 ORM 프레임워크는 원시 SQL 쿼리를 수행하기 위해 Manager.raw() 기능을 제공한다.

다음은 안전하지 않은 코드로 6라인에서 입력받은 외부 입력값을 10라인의 쿼리문 생성에 문자열 조합으로 사용하여 쿼리문의 구조를 만들고 있다.

안전하지 않은 코드의 예

```
1:  from django.shortcuts import render
2:  from app.models import Member
3:
4:  def member_search(request):
5:      # 외부로부터 입력 값을 가져온다.
6:      name = request.POST.get('name', '')
7:
8:      # 외부로부터 입력 받은 값을 검증 없이 쿼리문 생성에 사용하여
9:      # 안전하지 않다.
10:     query="select * from member where name='" + name + "'"
11:
12:     # 외부 입력 값을 검증 없이 사용한 쿼리문을 raw()함수로 실행하면
13:     # 안전하지 않다.
14:     data = Member.objects.raw(query)
15:     return render(request, '/member_list.html', {'member_list':data})
```

다음은 안전한 코드 예제로 Django에서 원시코드를 실행할 경우에도 매개변수화된 쿼리를 사용하고 params 인수를 사용하여 raw() 함수의 바인딩 변수로 사용하여야 한다. 9라인에서 쿼리문 생성을 매개변수화된 쿼리로 생성하고 6라인에서 입력받은 외부 입력값을 10라인의 raw()메소드에서 두 번째 인자의 바인딩변수로 사용하였다.

안전한 코드의 예

```
1:  from django.shortcuts import render
2:  from app.models import Member
3:
4:  def member_search(request):
5:      # 외부로부터 입력 값을 가져온다.
6:      name = request.POST.get('name', '')
7:
8:      # 외부 입력 값을 raw()함수 실행 시 바인딩 변수로 사용하여 쿼리 구조가
9:      # 변경되지 않도록 한다.(list 형은 %s, dictionary 형은 %(key)s를 사용)
10:     query='select * from member where name=%s'
11:
12:     # 인자화된 쿼리문을 사용하여 raw()함수를 사용하여 안전하다.
13:     data = Member.objects.raw(query, [name])
14:      return render(request, '/member_list.html', {'member_list':data})
```

라. 참고자료

① CWE-89: Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection'), MITRE, https://cwe.mitre.org/data/definitions/89.html

② SQL Injection Prevention Cheat Sheet, OWASP
https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html

③ Sqlite3 DB-API, Python, Python Software Foundation,
https://docs.python.org/ko/3/library/sqlite3.html

④ MySQL, Python Coding Examples, Oracle Corporation
https://dev.mysql.com/doc/connector-python/en/connector-python-examples.html

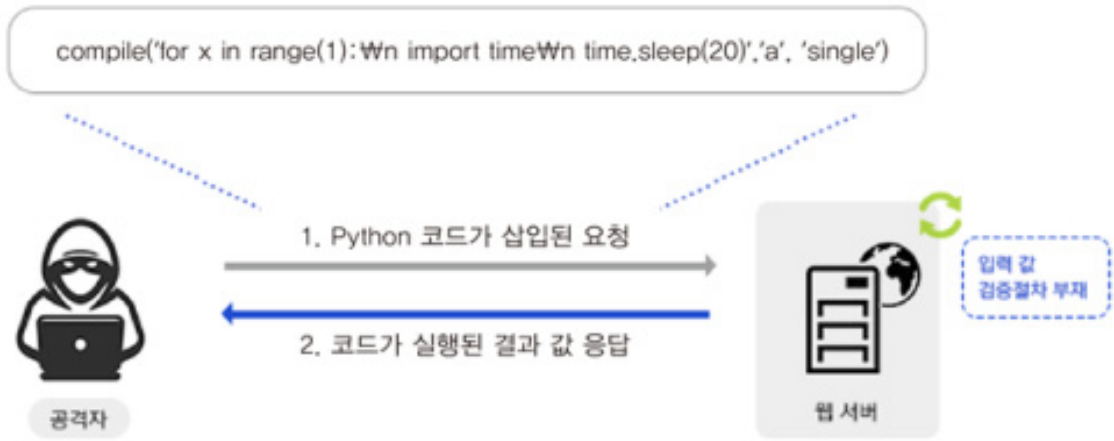
⑤ Django QuerySet API reference, Django Software Foundation,
https://docs.djangoproject.com/en/3.2/ref/models/querysets/

⑥ Django Performing raw SQL queries, Django Software Foundation,
https://docs.djangoproject.com/en/3.2/topics/db/sql/

⑦ SQL Expression Language Tutorial, SQLAlchemy,
https://docs.sqlalchemy.org/en/14/core/tutorial.html#using-textual-sql

2. 코드 삽입

가. 개요



공격자가 소프트웨어의 의도된 동작을 변경하도록 임의 코드를 삽입하여 소프트웨어가 비정상적으로 동작하도록 하는 보안 약점을 말한다. 코드 삽입은 프로그래밍 언어 자체의 기능에 한해 이뤄진다는 점에서 운영체제 명령어 삽입과 다르다.

취약한 프로그램에서 사용자의 입력 값에 코드가 포함되는 것을 허용할 경우, 공격자는 개발자가 의도하지 않은 코드를 실행하여 권한을 탈취하거나 인증 우회, 시스템 명령어 실행 등을 할 수 있다.

Python에서 코드 삽입 공격을 유발할 수 있는 함수로 eval(), exec()등의 함수가 있다. 해당 함수의 인자를 면밀히 검증하지 않는 경우 공격자가 전달한 코드를 그대로 실행할 수 있다.

나. 안전한 코딩기법

동적코드를 실행할 수 있는 함수를 사용하지 않는다. 필요 시, 실행 가능한 동적코드를 입력 값으로 받지 않도록, 외부 입력값에 대하여 화이트리스트 방식으로 검증한다. 또는 유효한 문자만 포함하도록 동적 코드에 사용되는 사용자 입력 값을 필터링 한다.

다. 코드예제

가) eval()함수 사용 예제

다음은 안전하지 않은 예제소스코드로 eval()을 사용하여 사용자로부터 입력받은 값을 실행하여 결과를 리턴하는 예제이다. 5라인에서 입력 받은 값을 아무런 검증 없이 10라인에서 eval()함수의 인자로 사용하고 있다. 외부 입력값을 검증 없이 사용할 경우 공격자는 Python 코드를 입력하여 라이브러리 로드 및 원격 대화형 쉘 등을 실행할 수도 있다. 만일 message에 공격자가 다음과 같은 코드를 입력 하면 20초 동안 응용프로그램이 sleep 상태에 빠질 수 있다.

예시) “compile(‘for x in range(1):\n import time\n time.sleep(20)’,’a’,’single’)”

안전하지 않은 코드의 예

```
1:  from django.shortcuts import render
2:
3:  def route(request):
4:      #외부로 입력받은 값을 검증 없이 사용하면 안전하지 않다.
5:      message = request.POST.get('message', '')
6:
7:      # eval함수에 외부 입력값을 검증 없이 사용할 경우 Python 코드가
8:      # 실행될 수 있어 위험하다.
9:
10:     ret = eval(message)
11:     return render(request, '/success.html', {'data':ret})
```

다음은 안전한 코드로 변환한 예제이다. 외부 입력값에 Python 코드를 실행할 수 있는 특수 문자 등을 필터링 하여 사전에 검증하는 코드를 추가하면 코드삽입을 완화할 수 있다. 아래 코드는 4라인에서 입력 받은 외부 입력값을 10라인의 eval() 함수의 인자 값으로 사용하기 전에 9라인에서 입력 값이 영문과 숫자만으로 입력되었는지 검증 후 사용하도록 하고 있다.

안전한 코드의 예

```
1:  from django.shortcuts import render
2:
3:  def route(request):
4:      message = request.POST.get('message', '')
5:
6:      # 사용자 입력을 영문, 숫자로 제한하는 예로. 특수문자가 포함되어
7:      # 있을 경우 에러 메시지를 리턴 한다.
8:
9:      if message.isalnum():
10:         ret = eval(message)
11:         return render(request, '/success.html', {'data':ret})
12:     ......
```

Python은 다양한 String Methods를 제공하고 있다. 외부 입력값에 대한 검증이 필요한 경우 적절한 메소드를 사용해야 한다. 아래는 Python에서 제공하는 String Methods중 일부 예시이다.

- str.isalpha() : 문자열 내의 모든 문자가 알파벳이고, 적어도 하나의 문자가 존재하는 경우 True를 반환.
- str.isdecimal() : 문자열 내의 모든 문자가 십진수 문자이고, 적어도 하나의 문자가 존재하는 경우 True를 반환.
- str.isdigit() : 문자열 내의 모든 문자가 숫자이고, 적어도 하나의 문자가 존재하는 경우 True를 반환. 십진수 문자와 호환되는 위 첨자 숫자와 같은 숫자도 포함. ex) '5^2' 는 True를 반환.

입력 값 검증 시 외부 입력값이 특정 형식을 따라야 할 경우 정규식을 이용하여 검증을 할 수 있고, Python의 re 라이브러리를 사용하여 검증 가능하다.

예로 Email 형식의 입력만 받아야 한다면 다음과 같은 정규식으로 검증 가능하다.

ex) prog = re.compile(r'([A-Za-z0-9]+[.-_])*[A-Za-z0-9]+@[A-Za-z0-9-]+(\.[A-Z|a-z]{2,})+')

나) exec()함수 사용 예제

다음은 exec()함수를 사용한 안전하지 않은 코드 예제이다. 4라인에서 입력받은 외부 입력값을 검증 없이 9라인의 exec 함수의 인자로 사용하고 있다. 이 경우 중요 데이터 탈취 및 서버의 권한 탈취, 액세스 거부, 심지어 완전한 호스트 탈취를 유발할 수 있다.

안전하지 않은 코드의 예

```
1:  from django.shortcuts import render
2:
3:  def request_rest_api(request):
4:      function_name = request.POST.get('function_name', '')
5:
6:      # 사용자에게 전달받은 함수명을 검증하지 않고 실행
7:      # 입력 값으로 "import plaform \n platform.system()" 등을 입력 시
8:      # 시스템정보 노출 위험이 있다.
9:      ret = exec('{}()'.format(function_name))
10:
11:     return render(request, '/success', {'data':ret})
```

다음은 안전한 코드로 변환한 예제이다. 우선 외부에서 입력 받는 라이브러리 명을 미리 정의한 화이트리스트에 있는 지를 확인하고 리스트에 없는 경우엔 에러 페이지를 리턴한다.

안전한 코드의 예

```
1:  from django.shortcuts import render
2:
3:  WHITE_LIST = ['get_friends_list', 'get_address', 'get_phone_number']
4:
5:  def request_rest_api(request):
6:      function_name = request.POST.get('function_name', '')
7:
8:      # 함수 명을 화이트리스트로 제한
9:      if function_name in WHITE_LIST:
10:         ret = exec('{}()'.format(function_name))
11:         return render(request, '/success', {'data':ret})
12:
13:     return render(request, '/error', {'error':'허용되지 않은 함수입니다.'})
```

라. 참고자료

① CWE-94: Improper Control of Generation of Code ('Code Injection'), MITRE,
https://cwe.mitre.org/data/definitions/94.html

② CWE-95: Improper Neutralization of Directives in Dynamically Evaluated Code ('Eval Injection'), MITRE,
https://cwe.mitre.org/data/definitions/95.html

③ Code Injection, OWASP,
https://owasp.org/www-community/attacks/Code_Injection

④ Python Built-in Functions – eval(), exec(), compile(), Python Software Foundation,
https://docs.python.org/3/library/functions.html#eval
https://docs.python.org/3/library/functions.html#exec
https://docs.python.org/3/library/functions.html#compile

⑤ Python Built-in Types – isalnum(), Python Software Foundation,
https://docs.python.org/3/library/stdtypes.html

⑥ Regular expression operations, Python Software Foundation,
https://docs.python.org/3/library/re.html#module-re

3. 경로 조작 및 자원 삽입

가. 개요

검증되지 않은 외부 입력값을 통해 파일 및 서버 등 시스템 자원(파일, 소켓의 포트 등)에 대한 접근 혹은 식별을 허용할 경우, 입력 값 조작을 통해 시스템이 보호하는 자원에 임의로 접근할 수 있는 보안약점이다. 경로조작 및 자원삽입 약점을 이용하여 공격자는 자원의 수정·삭제, 시스템 정보누출, 시스템 자원 간 충돌로 인한 서비스 장애 등을 유발시킬 수 있다.

즉, 경로 조작 및 자원 삽입을 통해서 공격자가 허용되지 않은 권한을 획득하여, 설정에 관계된 파일을 변경하거나 실행시킬 수 있다.

Python에서는 subprocess.popen()과 같이 프로세스를 여는 함수, os.pipe()처럼 파이프를 여는 함수, socket 연결 등에서 외부 입력값을 검증 없이 사용할 경우 경로 조작 및 자원 삽입의 취약점이 발생할 수 있다.

나. 안전한 코딩기법

외부의 입력 값을 자원(파일, 소켓의 포트 등)의 식별자로 사용하는 경우, 적절한 검증을 거치도록 하거나, 사전에 정의된 리스트에서 선택되도록 한다. 특히, 외부의 입력이 파일명인 경우에는 경로순회(directory traversal) 공격의 위험이 있는 문자(/, \, .. 등)를 제거할 수 있는 필터를 이용한다.

다. 코드예제

가) 경로 조작 예제

6라인에서 외부 입력값으로 파일 경로 등을 입력받아 14라인에서 파일을 오픈하고 있다. 만약 공격자가 ../../../../etc/passwd와 같은 값을 전달하면 의도하지 않았던 파일의 내용이 클라이언트 측에 표시되어 시스템 정보노출 문제가 발생한다.

안전하지 않은 코드의 예

```
1:  import os
2:  from django.shortcuts import render
3:
4:  def get_info(request):
5:      # 외부로부터 파일명을 입력받고 있다.
6:      request_file = request.POST.get('request_file')
7:      filename, file_ext = os.path.splitext(request_file)
8:      file_ext = file_ext.lower()
9:
10:     if file_ext not in ['.txt', '.csv']:
11:         return render(request, '/error.html', {'error':'파일을 열수 없습니다.'})
12:
13:     # 외부로부터 입력받은 값을 검증 없이 파일 처리에 사용하였다.
14:     with open(request_file) as f:
15:         data = f.read()
16:
17:     return render(request, '/success.html', {'data':data})
```

외부 입력값에서 경로 조작 문자열 (/, \, .. 등)을 제거한 후 파일의 경로설정에 사용한다.

안전한 코드의 예

```
1:  import os
2:  from django.shortcuts import render
3:
4:  def get_info(request):
5:      # 외부 입력 값으로 받은 파일 이름은 검증하여 사용한다.
6:      request_file = request.POST.get('request_file')
7:
8:      filename, file_ext = os.path.splitext(request_file)
9:      file_ext = file_ext.lower()
10:
11:     if file_ext not in ['.txt', '.csv']:
12:         return render(request, '/error.html', {'error':'파일을 열수 없습니다.'})
13:
14:     # 파일 명에서 경로 조작 문자열을 필터링 한다.
15:     filename = filename.replace('.', '')
16:     filename = filename.replace('/', '')
17:     filename = filename.replace('\\', '')
18:
19:     with open(filename + file_ext) as f:
21:         data = f.read()
22:
23:     return render(request, '/success.html', {'data':data})
```

relplace 함수 외에도 re.sub, filter 함수를 사용하여 특수문자를 필터링 하는 것도 가능하다.

나) 자원 삽입 예제

다음의 예제는 안전하지 않은 코드의 예로, 외부의 입력을 소켓 포트번호로 그대로 사용하고 있다. 외부 입력값을 검증없이 사용할 경우 기존 자원과 충돌로 에러가 발생할 수 있다.

안전하지 않은 코드의 예

```
1:  import socket
2:  from django.shortcuts import render
3:
4:  def get_info(request):
5:      port = request.POST.get('port')
6:
7:      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
8:          # 외부로부터 입력받은 검증되지 않은 포트번호를 이용하여
9:          # 소켓을 바인딩 하여 사용하고 있어 안전하지 않다.
10:         s.bind(('127.0.0.1', port))
11:         ...
12:         return render(request, '/success')
13:     return render(request, '/error', {'error':'소켓연결 실패'})
```

다음은 안전한 예제를 나타낸 것이다. 내부자원에 접근할 때 외부에서 입력 받은 값을 포트번호와 같은 식별자로 그대로 사용하는 것은 바람직하지 않으며, 꼭 필요한 경우엔 가능한 리스트를 설정하고, 해당 범위 내에서 할당되도록 작성한다.

안전한 코드의 예

```
1:  import socket
2:  from django.shortcuts import render
3:
4:  ALLOW_PORT = [4000, 6000, 9000]
5:
6:  def get_info(request):
7:      port = int(request.POST.get('port'))
8:
9:      # 사용 가능한 포트 번호를 화이트리스트로 제한
10:     if port not in ALLOW_PORT:
11:         return render(request, '/error', {'error':'소켓연결 실패'})
12:
13:     with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
14:         s.bind(('127.0.0.1', port))
15:         ......
16:     return render(request, '/success')
```

라. 참고자료

① CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal'), MITRE, https://cwe.mitre.org/data/definitions/22.html

② CWE-99: Improper Control of Resource Identifiers ('Resource Injection'), MITRE, https://cwe.mitre.org/data/definitions/99.html

③ Path Traversal, OWASP, https://owasp.org/www-community/attacks/Path_Traversal

④ Resource Injection, OWASP, https://owasp.org/www-community/attacks/Resource_Injection

⑤ File Uploads, Django Software Foundation, https://docs.djangoproject.com/en/3.2/topics/http/file-uploads/

⑥ HTML Helpers, Werkzeug, https://werkzeug.palletsprojects.com/en/2.0.x/utils/#module-werkzeug.utils

4. 크로스사이트 스크립트(XSS)

가. 개요

크로스사이트 스크립트 공격(Cross-site scripting Attacks)은 웹사이트에 악성 코드를 삽입하는 공격 방법이다. 공격자는 대상 웹 응용 프로그램의 결함을 이용하여 악성코드(일반적으로 클라이언트 측 JavaScript 사용)를 사용자에게 보낸다. XSS 공격은 일반적으로 애플리케이션의 호스트 자체를 대상으로 하지 않고 애플리케이션의 사용자를 목표로 삼는다.

XSS는 공격자가 웹 응용프로그램을 속여 사용자의 브라우저에서 실행할 수 있는 형식의 데이터를 보낼 때 발생한다. 일반적인 HTML과 공격자가 제공한 XSS코드의 조합 뿐만 아니라 악성코드 다운로드, 플러그인 또는 미디어 콘텐츠를 이용하기도 한다. 사용자가 양식에 입력한 데이터 또는 서버에서 클라이언트 소프트웨어(브라우저 또는 WebKit등)의 종료점(endpoint)으로 전달된 데이터가 적절한 검증 없이 사용자에게 표시되도록 허용하는 경우 발생한다.

XSS공격은 크게 세가지 유형의 공격방법이 있다.

• 유형 1 : Reflective XSS (or Non-persistent XSS)

- Reflective XSS는 공격 코드를 사용자의 HTTP 요청에 삽입시킨 뒤, 해당 공격 코드가 서버 응답 내용에 그대로 반사(Reflected)되어 브라우저에서 실행시키는 공격기법이다. Reflective XSS를 악용하려면 공격자가 사용자를 속여 대상 사이트로 데이터를 보내도록 해야 한다. 이 방법은 종종 사용자가 악의적으로 제작된 링크를 클릭하도록 속임으로써 수행된다. 대부분의 경우 Reflective XSS공격 메커니즘은 공개적으로 게시되거나 피해자 전송되는 피싱(Phishing) 이메일 또는 단축 URL 또는 모호한 URL에 매개 변수로 포함하는 방법이다. 이러한 방식으로 구성된 URL은 많은 피싱 체계의 핵심을 구성하며, 공격자는 피해자가 취약한 사이트를 참조하는 URL을 방문하도록 유도한다. 피해자가 링크를 방문하면 스크립트가 피해자의 브라우저에서 자동으로 실행됩니다.

• 유형 2 : Persistent XSS (or Stored XSS)

- Persistent XSS는 신뢰할 수 없거나 확인되지 않은 사용자 입력이 대상 서버에 저장될 때 발생한다. Persistent XSS의 일반적인 대상에는 게시판 글, 댓글 또는 방문자 로그가 포함된다. 이 기능들은 인증되거나 인증되지 않은 다른 사용자가 공격자의 악성 콘텐츠를 볼 수 있는 기능이다. 소셜(Social) 미디어 사이트 및 회원 그룹에서 흔히 볼 수 있는 것과 같이 공개적으로 표시되는 프로필 페이지는 Persistent XSS의 손쉬운 공격 대상 중 하나입니다. 공격자는 프로필 입력박스에 악성 스크립트를 입력할 수 있으며 다른 사용자가 프로필을 방문하면 브라우저에서 자동으로 코드가 실행되도록 할 수 있다.

• **유형 3 : DOM 기반 XSS (or Client-Side XSS)**

- DOM 기반 XSS은 공격 스크립트가 포함된 악성 URL을 통해 전달되는 경우가 많기 때문에 Reflective XSS와 어느 정도 유사하다. 그러나 신뢰할 수 있는 사이트의 HTTP 응답에 페이로드를 포함하는 대신 DOM 또는 문서 개체 모델을 수정하여 브라우저에서 독립적으로 완전한 공격을 실행한다. 이는 웹 페이지에 있는 사용자 입력 값을 적절하게 처리하기 위한 JavaScript의 검증 로직을 무효화 하는 것을 목표로 한다.

  악성 스크립트가 삽입되면 공격자는 다양한 악성 활동을 수행 할 수 있다. 공격자는 세션 정보를 포함 한 쿠키와 같은 개인 정보를 피해자의 컴퓨터에서 공격자에게 전송할 수 있다. 공격자는 피해자의 정로를 사용하여 웹 사이트에 악의적 인 요청을 보낼 수 있으며, 피해자가 해당 사이트를 관리 할 수 있는 관리자 권한이 있는 경우 사이트에 특히 위험할 수 있다. 피싱(Phishing) 공격은 신뢰할 수 있는 웹 사이트를 모방하고 피해자가 암호를 입력하도록 속여 공격자가 해당 웹 사이트에서 피해자의 계정을 손상시킬 수 있다.

  파이썬에서 가장 많이 사용하고 있는 Django 프레임워크와 Flask 프레임워크에서는 각각 Django 템플릿과 Jinja2 템플릿을 사용할 시 크로스사이트 스크립트 공격에 사용될 수 있는 위험한 HTML 문자에 대해 HTML 특수문자(HTML Entities)로 치환하는 기능을 제공하고 있어 프레임워크에서 제공하는 템플릿을 사용하는 경우 안전하게 사용할 수 있다.

나. 안전한 코딩기법

외부 입력값 또는 출력값에 스크립트가 삽입되지 못하도록 문자열 치환 함수를 사용하여 &<>*‘/()등을 & < > " ' / ( )로 치환하거나, html라이브러리의 escape()를 사용한다. HTML 태그를 허용하는 게시판에서는 허용되는 HTML 태그들을 화이트리스트로 만들어 해당 태그만 지원하도록 한다.

파이썬에서 가장 많이 사용하는 프레임워크인 Django, Flask등을 사용하는 경우 외부 입력값에 악의적인 스크립트가 삽입되지 못하도록 프레임워크 자체에서 크로스사이트 스크립트 공격에 사용될 수 있는 문자를 HTML 특수문자(HTML Entities)로 치환하여 응답 페이지를 생성하므로 크로스사이트 스크립트로부터 보호된다. 프레임워크에서 크로스사이트 스크립트 공격으로부터 보호해 주는 기능이 있더라도 완전하지 않은 경우도 있고 개발자의 실수로 보호기능이 무효화 되는 경우가 있어 주의를 기울여야 한다.

다. 코드예제

가) Django 예제

Django 프레임워크는 크로스사이트 스크립트 공격에 대한 보안기능을 내장하고 있지만 유의해야 할 사항이 몇 가지 있다. Django의 “safestring(django.utils.safestring)”의 기능을 오용할 경우 Django의 크로스사이트 스크립트 보호 정책이 무력화 될 수 있다.

안전하지 않은 코드의 예

```
1:  from django.shortcuts import render
2:  from django.utils.safestring import mark_safe
3:
4:  def profile_link(request):
5:      # 외부 입력값을 검증 없이 HTML 태그 생성의 인자로 사용
6:      profile_url = request.POST.get('profile_url')
7:      profile_name = requst.POST.get('profile_name')
8:
9:      object_link = '<a href="{}">{}</a>'.format(profile_url, profile_name)
10:     # mark_safe함수는 Django의 XSS escape 정책을 따르지 않는다.
11:     object_link = mark_safe(object_link)
12:
13:     return render(request, 'my_profile.html',{'object_link':object_link})
```

Django 프레임워크는 템플릿 생성 시 HTML에서 위험한 것으로 간주되는 특수 문자(“<”, “>”, “‘”, “””, “&”)를 모두 html 엔티티로 치환 하지만 mark_safe를 사용할 경우 이 정책을 따르지 않는다.

따라서 mark_safe 함수를 사용 할 경우에는 각별한 주의가 필요하고 신뢰할 수 없는 데이터에 대해서는 mark_safe 함수를 사용하지 않는다.

안전한 코드의 예

```
1:  from django.shortcuts import render
2:
3:  def profile_link(request):
4:      # 외부 입력값을 검증 없이 HTML 태그 생성의 인자로 사용
5:      profile_url = request.POST.get('profile_url')
6:      profile_name = requst.POST.get('profile_name')
7:
8:      object_link = '<a href="{}">{}</a>'.format(profile_url, profile_name)
9:      # 신뢰 할 수 없는 데이터에 대해서는 mark_safe 함수를 사용하지 않는다
10:
11:     return render(request, 'my_profile.html',{'object_link':object_link})
```

다음은 Django 프레임워크에서의 템플릿 예제이다. autoescape 블록을 사용 시 설정값을 off로 할 경우와 개별 변수에 대해서 safe 필터를 사용할 경우 크로스사이트 스크립트 공격에 노출될 수 있다.

안전하지 않은 코드의 예

```
1:  <!doctype html>
2:  <html>
3:      <body>
4:          <div class="content">
5:              {% autoescape off %}
6:                  // autoescape off로 해당 블록내의 데이터는 XSS 공격에
7:                  // 노출될 수 있음.
8:                  {{ content }}
9:              {% endautoescape %}
10:         </div>
11:         <div class="content2">
12:                 //safe 필터 사용으로 XSS 공격에 노출될 수 있음.
13:                 {{ content | safe }}
14:         </div>
15:     </body>
16: </html>
```

신뢰할 수 없는 입력값 또는 동적 데이터에 대해서는 autoescape 옵션 값을 on으로 하고 safe 필터를 부득이 하게 사용할 경우 추가적인 보안대책이 필요하다.

안전한 코드의 예

```
1:  <!doctype html>
2:  <html>
3:      <body>
4:          <div class="content">
5:              {% autoescape on %}
6:                  // autoescape on로 해당 블록내의 데이터는 XSS 공격에
7:                  {{ content }}
8:              {% endautoescape %}
9:          </div>
10:         <div class="content2">
11:                 //검증되지 않은 데이터에는 safe 필터를 사용하지 않는다.
12:                 {{ content }}
13:         </div>
14:     </body>
15: </html>
```

autoescape 블록을 사용할 경우 많은 주의를 기울여야 한다. autoescape 옵션값을 off로 설정한 템플릿 페이지를 include 또는 extends하는 템플릿까지 영향이 확장된다. 공통적으로 사용하는 템플릿페이지에 off로 설정할 경우 많은 템플릿 페이지가 크로스사이트 스크립트 공격에 노출될 수 있다.

나) Flask에서의 예제

사용자의 요청에 포함된 값 또는 DB에 저장된 값 또는 내부의 연산을 통해서 생성된 값을 포함 한 값은 동적 웹 페이지 생성에 사용하는 경우 크로스사이트 스크립트(XSS) 공격이 발생할 가능성이 있어 위험하다. 아래 예제는 Flask 프레임워크를 사용한 안전하지 않은 예제이다.

안전하지 않은 코드의 예

```
1:  from flask import Flask, request, render_template
2:
3:  @app.route('/search', methods=['POST'])
4:  def search():
5:      search_keyword = request.form.get('search_keyword')
6:      # 사용자의 입력을 아무런 검증 또는 치환 없이 동적 웹 페이지에 사용하고 있어
7:      # 크로스 사이트 스크립트가 발생할 수 있다
8:      return render_template('search.html', search_keyword=search_keyword)
```

동적 웹 페이지 생성에 사용되는 데이터를 HTML 엔티티 코드 (Entity Code)로 치환하여 안전하게 표현하여야 한다. html.escape 메소드는 문자열의 &, 〈 및 〉 특수문자를 HTML에서 안전한 값으로 변환한다. 옵션 값의 quote 값이 True이면 문자 (“)와 (‘)도 변환된다. 〈a href=”…“〉에서처럼 따옴표로 구분된 HTML 어트리뷰트 값에 포함하는 문자열을 포함할 경우 사용할 수 있다.

안전한 코드의 예

```
1:  import html
2:  from flask import Flask, request, render_template
3:
4:  @app.route('/search', methods=['POST'])
5:  def search():
6:      search_keyword = request.form.get('search_keyword')
7:
8:      # 동적 웹 페이지 생성에 사용되는 데이터는
9:      # HTML 엔티티코드로 치환하여 표현해야한다
10:     escape_keyword = html.escape(search_keyword)
11:     return render_template('search.html', search_keyword=escape_keyword)
```

라. 참고자료

① CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting'), MITRE, https://cwe.mitre.org/data/definitions/79.html

② Cross Site Scripting (XSS), OWASP, https://owasp.org/www-community/attacks/xss/

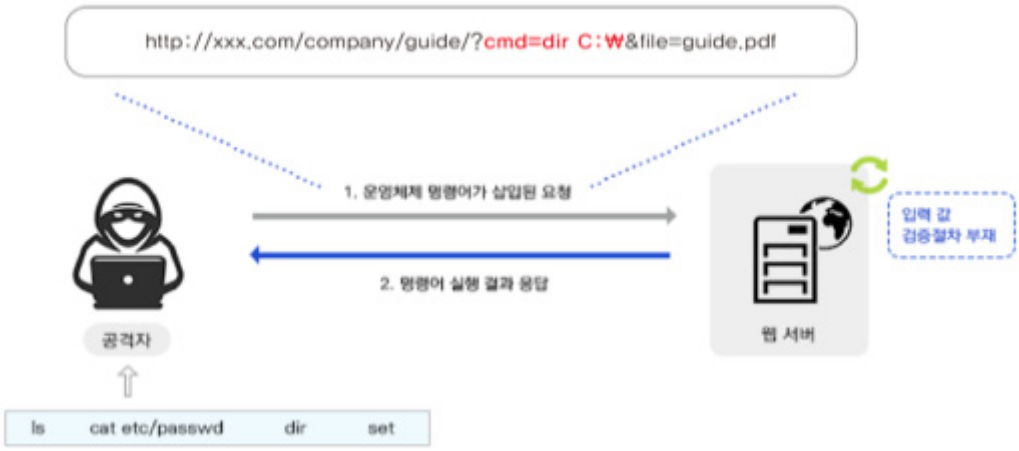
③ html - HyperText Markup Language support, Python Software Foundation, https://docs.python.org/3/library/html.html

④ Flask Security Considerations Cross-Site Scripting (XSS), Flask docs, https://flask-docs-kr.readthedocs.io/ko/latest/security.html

⑤ Django Security in Django Cross site scripting (XSS) protection, Django Software Foundation, https://docs.djangoproject.com/en/3.2/topics/security/

5. 운영체제 명령어 삽입

가. 개요



적절한 검증절차를 거치지 않은 사용자 입력값이 운영체제 명령어의 일부 또는 전부로 구성되어 실행되는 경우, 의도하지 않은 시스템 명령어가 실행되어 부적절하게 권한이 변경되거나 시스템 동작 및 운영에 악영향을 미칠 수 있다.

일반적으로 명령어 라인의 파라미터나 스트림 입력 등 외부 입력을 사용하여 시스템 명령어를 생성 하는 프로그램이 많이 있다. 하지만 이러한 경우 외부 입력 문자열은 신뢰할 수 없기 때문에 적절한 처리를 해주지 않으면, 공격자가 원하는 명령어 실행이 가능하게 된다.

파이썬에서 eval() 함수와 exec() 함수는 내부에서 문자열을 실행하기에 편리하지만, String 형식의 표현된 식을 인수로 받아 반환하는 eval() 함수와 인수로 받은 문자열을 실행하는 exec()를 같이 사용하면 여러 변수들에 동적으로 값을 할당하여 사용될 수 있어 Command Injection 공격에 취약하다.

나. 안전한 코딩기법

외부 입력값이 시스템 명령에 포함되는 경우 |, ;, &, :, >, <, `(backtick), \, ! 과 같이 멀티라인 파이프 리다이렉트 문자 등을 필터링 하고 명령을 수행할 파일명, 옵션을 제한하여 인자로만 사용될 수 있도록 한다. 외부 입력에 따라 명령어를 생성하거나 선택이 필요한 경우에는 명령어 생성에 필요한 값 들을 미리 지정해 놓고 외부 입력에 따라 선택하여 사용한다.

다. 코드예제

다음 예제는 os.system을 이용하여 외부 입력 받은 값을 통해 프로그램을 실행하며, 외부에서 전달되는 인자 값은 명령어의 생성에 사용된다. 그러나 해당 프로그램에서 실행할 프로그램을 제한하지 않고 있기 때문에 외부의 공격자는 가능한 모든 프로그램을 실행시킬 수 있다.

안전하지 않은 코드의 예

```
1:  import os
2:  from django.shortcuts import render
3:
4:  def execute_command(request):
5:      app_name_string = request.POST.get('app_name','')
6:      # 입력받은 파라미터를 제한하지 않아 외부 입력값으로 전달된
7:     # 모든 프로그램이 실행될 수 있음
8:      os.system(app_name_string)
9:      return render(request, '/success.html')
```

외부에서 입력 받은 값이 명령어의 인자로 사용되지 않고 명령어로 사용될 경우에는 미리 화이트리스트의 정의된 파라미터 배열을 만들어 놓고, 외부의 입력에 따라 적절한 파라미터를 선택하도록 하여, 외부의 부적절한 입력이 명령어로 사용될 가능성을 배제하여야 한다.

안전한 코드의 예

```
1:  import os
2:  from django.shortcuts import render
3:
4:  ALLOW_PROGRAM = ['notepad', 'calc']
5:
6:  def execute_command(request):
7:      app_name_string = request.POST.get('app_name','')
8:
9:      # 입력받은 파라미터를 사용가능한 시스템 명령어 일부로 제한하여 사용
10:     if app_name_string not in ALLOW_PROGRAM:
11:         return render(request, '/error.html', {'ERROR':'허용되지 않은 프로그램입니다.'})
12:
13:     os.system(app_name_string)
14:     return render(request, '/success.html')
```

다음은 subprocess()함수를 사용하여 별도의 프로세스로 응용프로그램을 실행하는 경우의 안전하지 않은 예제이다. 외부 입력값으로 받은 파라미터를 별도의 검증 없이 subprocess의 인자 값으로 사용하고 있다.

안전하지 않은 코드의 예

```
1:  import subprocess
2:  from django.shortcuts import render
3:
4:  def execute_command(request):
5:      date = request.POST.get('date','')
6:      # 입력받은 파라미터를 제한하지 않아 전달된 모든 프로그램이
7:      # 실행될 수 있음
8:      cmd_str = "cmd /c backuplog.bat " + date
9:      subprocess.run(cmd_str, shell=True)
10:     return render(request, '/success.html')
```

운영체제 명령어 실행 시에는 아래와 같이 외부에서 들어오는 값에 의하여 멀티라인을 지원하는 특수문자(|, ;, &, :, `, \, !)나 파일 리다이렉트 특수문자(>, >>)등을 제거하여 원하지 않는 운영체제 명령어가 실행 될 수 없도록 필터링을 수행한다.

Command lines을 구문 분석하고 escape하는 기능을 제공하는 모듈인 shlex 모듈을 사용하여 필터링을 수행한다. subprocess의 옵션 값 중 shell=True 일 경우 중간 프로세스에 의해 명령이 실행되고 파일 이름, 와일드카드(*), 환경변수 확장 등의 쉘 기능을 검증 없이 실행하게 되므로 shell의 옵션은 삭제 한다. (default 값은 False)

안전한 코드의 예

```
1:  import subprocess
2:  from django.shortcuts import render
3:
4:  def execute_command(request):
5:      date = request.POST.get('date','')
6:
7:      # 명령어를 추가로 실행 또는 다른 명령이 실행될 수 있는 키워드에
8:      # 대한 예외처리
9:      for word in ['|', ';', '&', ':', '>', '<', '"', '\\', '!']:
10:         date = date.replace(word, "")
11:
12:     # shell=True 옵션은 제거 하고 명령과 인자를 배열로 입력
13:     subprocess.run(["cmd", "/c", "backuplog.bat", date])
14:     return render(request, '/success.html')
```

re.sub함수를 사용하여 다음과 같이 특수문자를 제거할 수도 있다.

※ date = re.sub('[|;&:><\'\\\!]', '', date)

라. 참고자료

① CWE-78: Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection'), MITRE, https://cwe.mitre.org/data/definitions/78.html

② Command Injection, OWASP
https://owasp.org/www-community/attacks/Command_Injection

③ OS Command Injection Defense Cheat Sheet, OWASP
https://cheatsheetseries.owasp.org/cheatsheets/OS_Command_Injection_Defense_Cheat_Sheet.html

④ Miscellaneous operating system interfaces – os.system(), Python Software Foundation
https://docs.python.org/3.10/library/os.html?highlight=os%20system#module-os

⑤ Subprocess management, Python Software Foundation,
https://docs.python.org/ko/3/library/subprocess.html#security-considerations

⑥ Regular expression operations, Python Software Foundation,
https://docs.python.org/3/library/re.html

6. 위험한 형식 파일 업로드

가. 개요

서버 측에서 실행될 수 있는 스크립트 파일(asp, jsp, php, sh 파일 등)이 업로드 가능하고, 이 파일을 공격자가 웹을 통해 직접 실행시킬 수 있는 경우, 시스템 내부명령어를 실행하거나 외부와 연결하여 시스템을 제어할 수 있는 보안약점이다.

공격자가 실행 가능한 파일을 서버에 업로드 하면 파이썬에서 String 형식으로 표현된 표현식을 인수로 받아 반환하는 eval() 함수와 인수로 받은 문자열을 실행하는 exec()를 같이 사용하여 여러 변수들을 동적으로 값을 할당받아 실행될 수 있어 웹쉘(Web Shell) 공격에 취약하다.

나. 안전한 코딩기법

파일 업로드 공격을 방지하기 위해서 특정 파일 유형만 허용하도록 화이트리스트 방식으로 파일 유형을 제한하여야 한다. 이때 파일의 확장자 및 업로드 된 파일의 Content-Type도 같이 확인한다. 파일 크기 및 파일 개수를 제한하여 시스템 자원 고갈 등으로 서비스 거부 공격이 발생하지 않도록 제한하여야 한다. 업로드 된 파일을 웹 루트 폴더 외부에 저장하여 공격자가 URL을 통해 파일을 실행할 수 없도록 해야 하고 가능하면 업로드 되는 파일의 이름은 공격자가 추측할 수 없는 무작위한 이름으로 변경하여 저장하는 것이 안전하다. 또한 업로드 된 파일을 저장할 경우에는 최소 권한만 부여하는 것이 안전하고 실행 여부를 확인하여 실행권한을 삭제 한다.

다. 코드예제

업로드 할 파일에 대한 개수, 크기, 확장자 등의 유효성 검사를 하지 않고 파일시스템에 저장 할 경우 공격자에 의해 악성코드, 쉘 코드 등 위험한 형식의 파일을 시스템에 업로드 할 수 있다.

안전하지 않은 코드의 예

```
1:  from django.shortcuts import render
2:  from django.core.files.storage import FileSystemStorage
3:
4:  def file_upload(request):
5:      if request.FILES['upload_file']:
6:          # 사용자로부터 업로드 되는 파일에 대해 검증 없이 저장하고 있어
7:          # 안전하지 않다.
8:          upload_file = request.FILES['upload_file']
9:          fs = FileSystemStorage(location='media/screenshot', base_url='media/screenshot')
10:         # 업로드 하는 파일에 대한 크기, 개수, 확장자 등을 검증 하지 않음.
11:         filename = fs.save(upload_file.name, upload_file)
12:         return render(request, '/success.html', {'filename':filename})
13:     return render(request, '/error.html', {'error':'파일 업로드 실패'})
```

아래 코드는 업로드 하는 파일의 파일 개수, 파일 크기, 파일 확장자 등을 검사하여 업로드를 제한하고 있다. 21라인의 파일 타입 확인은 MIME 타입을 확인하는 과정으로 파일이름에서 확장자만 검사할 경우 변조된 확장자를 통해 업로드 제한을 회피할 수 있어 파일자체의 시그니처를 확인하는 과정이다.

안전한 코드의 예

```
1:  import os
2:  from django.shortcuts import render
3:  from django.core.files.storage import FileSystemStorage
4:
5:  # 업로드 하는 파일에 대한 개수, 크기, 확장자 제한
6:  FILE_COUNT_LIMIT = 5
7:  # 업로드 하는 파일의 최대 사이즈 제한 예 ) 5MB - 5*1024*1024
8:  FILE_SIZE_LIMIT = 5242880
9:  # 허용하는 확장자는 화이트리스트로 관리한다.
10: WHITE_LIST_EXT = [
11:     '.jpg',
12:     '.jpeg'
13: ]
14:
15: def file_upload(request):
16:     # 파일 개수 제한
17:     if len(request.FILES) == 0 or len(request.FILES) > FILE_COUNT_LIMIT:
18:         return render(request, '/error.html', {'error': '파일 개수 초과'})
19:
20:     for filename, upload_file in request.FILES.items():
21:         # 파일 타입 체크
```

안전한 코드의 예

```
22:         if upload_file.content_type != 'image/jpeg':
23:             return render(request, '/error.html', {'error': '파일 타입 오류'})
24:         # 파일 크기 제한
25:         if upload_file.size > FILE_SIZE_LIMIT:
26:             return render(request, '/error.html', {'error': '파일사이즈 오류'})
27:         # 파일 확장자 검사
28:         file_name, file_ext = os.path.splitext(upload_file.name)
29:         if file_ext.lower() not in WHITE_LIST_EXT:
30:             return render(request, '/error.html', {'error': '파일 타입 오류'})
31:
32:     fs = FileSystemStorage(location='media/screenshot', base_url = 'media/screenshot')
33:     for upload_file in request.FILES.values():
34:         fs.save(upload_file.name, upload_file)
35:
36:     return render(request, '/success.html',  {'filename': filename})
```

라. 참고자료

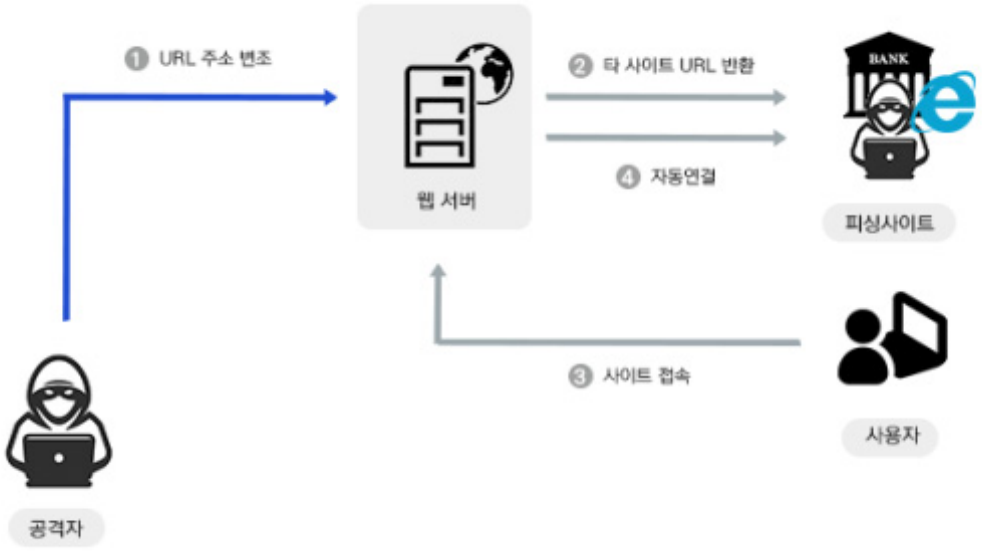
① CWE-434: Unrestricted Upload of File with Dangerous Type, MITRE, https://cwe.mitre.org/data/definitions/434.html

② Unrestricted File Upload, OWASP, https://owasp.org/www-community/vulnerabilities/Unrestricted_File_Upload

③ User-uploaded content, Django Software Foundation, https://docs.djangoproject.com/en/3.2/topics/security/#user-uploaded-content-security

7. 신뢰되지 않은 URL주소로 자동접속 연결

가. 개요



사용자로부터 입력되는 값을 외부사이트의 주소로 사용하여 자동으로 연결하는 서버 프로그램은 피싱(Phishing) 공격에 노출되는 취약점을 가질 수 있다.

일반적으로 클라이언트에서 전송된 URL 주소로 연결하기 때문에 안전하다고 생각할 수 있으나, 공격자는 해당 폼의 요청을 변조함으로써 사용자가 위험한 URL로 접속할 수 있도록 공격할 수 있다

Python 프레임워크의 redirect 함수를 사용 할 때에도 해당 프레임워크 버전에서 알려진 취약점이 있는지 확인해야 한다. Flask 프레임워크의 Flask-Security-Too 라이브러리의 경우 get_post_logout_redirect 함수와 get_post_login_redirect 함수가 4.1.0 이전 버전에서 URL 유효성 검사를 우회하고 사용자를 임의의 URL로 리디렉션 할 수 있는 취약점이 존재한다.

나. 안전한 코딩기법

리디렉션을 허용하는 모든 URL을 서버 측 화이트리스트로 관리하고 사용자 입력 값을 리디렉션 할 URL이 존재하는지 검증해야 한다.

만약 사용자 입력 값이 화이트 리스트로 관리가 불가능하고 리디렉션 URL의 인자 값으로 사용되어야만 하는 경우는 모든 리디렉션에서 프로토콜과 host정보가 들어가지 않는 상대 URL(relative)을 사용해야 하고, 검증해야 한다. 또는 절대 URL(absoute URL)을 사용할 경우 리디렉션을 실행하기 전에 사용자 입력 URL이 https://myhompage.com/ 처럼 서비스하고 있는 URL로 시작하는지를 확인해야 한다.

다. 코드예제

다음은 안전하지 않은 예제로 사용자로부터 입력 받은 url 주소를 검증 없이 reirect함수의 인자로 사용하고 있다. 이 경우 사용자가 의도하지 않은 사이트로 접근하도록 하거나 피싱(Phishing)공격에 노출될 수 있다.

안전하지 않은 코드의 예

```
1:  from django.shortcuts import redirect
2:
3:  def redirect_url(request):
4:      url_string = request.POST.get('url', '')
5:      # 사용자 입력에 포함된 URL 주소로 리다이렉트 하는 경우
6:      # 피싱 사이트로 접속되는 등 사용자가 피싱 공격에 노출될 수 있다
7:      return redirect(url_string)
```

다음은 안전한 코드 예제로 사용자로부터 주소를 입력받아 redirect하고 있는 코드로 위험한 도메인이 포함되어 있을 수 있기 때문에 화이트 리스트로 사전에 정의된 안전한 웹사이트에 한하여 redirect 할 수 있도록 한다.

안전한 코드의 예

```
1:  from django.shortcuts import render, redirect
2:
3:  ALLOW_URL_LIST = [
4:      '127.0.0.1',
5:      '192.168.0.1',
6:      '192.168.0.100',
7:      'https://login.myservice.com',
8:      '/notice'
9:  ]
10:
11: def redirect_url(request):
12:     url_string = request.POST.get('url', '')
13:
14:     # 이동할 수 있는 URL 범위를 제한하여
15:     # 위험한 사이트의 접근을 차단하고 있다
16:     if url_string not in ALLOW_URL_LIST:
17:         return render(request, '/error.html', {'ERROR':'허용되지 않는 주소입니다.})
18:
19:     return redirect(url_string)
```

라. 참고자료

① CWE-601: URL Redirection to Untrusted Site ('Open Redirect'), MITRE, https://cwe.mitre.org/data/definitions/601.html

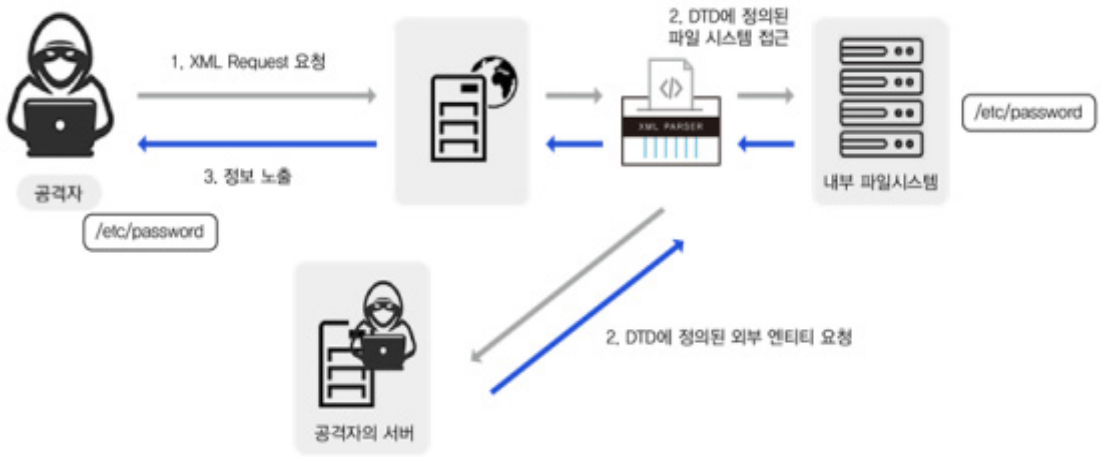
② Unvalidated Redirects and Forwards Cheat Sheet, OWASP https://cheatsheetseries.owasp.org/cheatsheets/Unvalidated_Redirects_and_Forwards_Cheat_Sheet.html

③ Django shortcut functions – redirect, Django Sowftware Foundation, https://docs.djangoproject.com/en/3.2/topics/http/shortcuts/

④ Redirects and Errors, Flask, https://flask.palletsprojects.com/en/2.0.x/quickstart/#redirects-and-errors

8. 부적절한 XML 외부 개체 참조

가. 개요



XML 문서에는 DTD(DocumentTypeDefinition)를 포함할 수 있으며, DTD는 XML 엔티티(entitiy)를 정의한다. 부적절한 XML 외부개체 참조 보안약점은 서버에서 XML 외부엔티티를 처리할 수 있도록 설정된 경우에 발생할 수 있다.

취약한 XML parser가 외부 값을 참조하는 XML 값을 처리할 때, 공격자가 삽입한 공격 구문이 동작되어 서버 파일 접근, 불필요한 자원 사용, 인증 우회, 정보 노출 등이 발생할 수 있다.

Python에서는 간단한 XML 데이터 구문 분석 및 조작에 사용할 수 있는 기본 XML 파서가 제공된다. 이 파서는 유효성 검사와 같은 고급 XML 기능은 지원하지 않는다. 기본 제공되는 XML 파서는 외부 엔티티를 지원하지 않지만 다른 XML 공격에 취약할 수 있다.

기본 제공되는 파서의 기능 외에 더 많은 기능이 필요한 경우에 lxml과 같은 라이브러리를 사용하게 되는데 이 라이브러리에서는 기본적으로 외부 엔티티의 구문 분석이 활성화 되어 있다.

나. 안전한 코딩기법

로컬 정적 DTD를 사용하도록 설정하고, 외부에서 전송된 XML 문서에 포함된 DTD를 완전하게 비활성화해야 한다. 비활성화를 할 수 없는 경우에는 외부 엔티티 및 외부 문서 유형 선언을 각 파서에 맞는 고유한 방식으로 비활성화 한다.

외부 라이브러리를 사용할 경우 기본적으로 외부 엔티티에 대한 구문 분석 기능을 제공하는지 확인하고 제공할 경우 해당 기능을 비활성화 할 수 있는 방법을 확인하여 외부 엔티티 구문 분석 기능을 비활성화 한다.

많이 사용하는 XML 파서의 한 종류인 lxml의 경우 외부 엔티티 구문 분석 옵션인 resolve_entities 옵션을 비활성화 하여야 한다. 또한 외부 문서를 조회할 때 네트워크 액세스를 방지하는 no_network 옵션이 활성화(True) 되어 있는지도 확인 하여야 한다.

다. 코드예제

다음 예제는 XML 소스를 읽어서 분석하는 코드이다. 공격자는 아래와 같이 XML 외부 엔티티를 참조하는 xxe.xml 데이터를 전송하고, 이를 파싱 할 때 /etc/passwd 파일을 참조할 수 있다.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
    <!ELEMENT foo ANY >
    <!ENTITY xxe1 SYSTEM "file:///etc/passwd" >
    <!ENTITY xxe2 SYSTEM "http://attacker.com/text.txt">
]>
<foo>&xxe1;&xxe2;</foo>
```

안전하지 않은 코드의 예

```
1:  from xml.sax import make_parser
2:  from xml.sax.handler import feature_external_ges
3:  from xml.dom.pulldom import parseString, START_ELEMENT
4:  from django.shortcuts import render
5:  from .model import comments
6:
7:  def get_xml(request):
8:      if request.method == “GET”:
9:          data = comments.objects.all()
10:         com = data[0].comment
11:         return render(request, '/xml_view.html', {'com':com})
12:
13:     elif request.method == “POST”:
14:         parser = make_parser()
15:         parser.setFeature(feature_external_ges, True)
16:         doc = parseString(request.body.decode(‘utf-8’), parser=parser)
17:         for event, node in doc:
18:             if event == START_ELEMENT and node.tagName == “foo”:
19:                 doc.expandNode(node)
20:                 text = node.toxml()
21:         comments.objects.filter(id=1).update(comment=text);
22:         return render(request, '/xml_view.html')
```

XML을 파싱 할 때 텍스트 값을 entities로 변환해 주는 옵션인 resolve_entities 기능을 False(default: True)로 설정해야 한다. 또한 연결된 파일의 네트워크 연결을 막아주는 no_network 값을 True(default: True)로 설정해야 한다.

안전한 코드의 예

```
1:  from xml.sax import make_parser
2:  from xml.sax.handler import feature_external_ges
3:  from xml.dom.pulldom import parseString, START_ELEMENT
4:  from django.shortcuts import render
5:  from .model import comments
6:
7:  def get_xml(request):
8:      if request.method == "GET":
9:          data = comments.objects.all()
10:         com = data[0].comment
11:         return render(request, '/xml_view.html', {'com':com})
12:
13:     elif request.method == "POST":
14:         parser = make_parser()
15:         parser.setFeature(feature_external_ges, False)
16:         doc = parseString(request.body.decode('utf-8'), parser=parser)
17:         for event, node in doc:
18:             if event == START_ELEMENT and node.tagName == "foo":
19:                 doc.expandNode(node)
20:                 text = node.toxml()
21:         comments.objects.filter(id=1).update(comment=text);
22:         return render(request, '/xml_view.html')
```

라. 참고자료

① CWE-611: Improper Restriction of XML External Entity Reference, MITRE, https://cwe.mitre.org/data/definitions/611.html

② XML External Entity (XXE) Processing, OWASP, https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_ Processing

③ XML External Entity Prevention Cheat Sheet, OWASP, https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html

④ XML vulnerabilities, Python Software Foundation, https://docs.python.org/3/library/xml.html#xml-vulnerabilities

⑤ lxml API, lxml library, https://lxml.de/api/lxml.etree.XMLParser-class.html

⑥ PyGoat, OWASP, https://github.com/adeyosemanputra/pygoat

9. XML 삽입

가. 개요

검증되지 않은 외부 입력값이 XQuery 또는 XPath 쿼리문을 생성하는 문자열로 사용되어 공격자가 쿼리문의 구조로 임의로 변경하고 임의의 쿼리를 실행하여 허가되지 않은 데이터를 열람하거나 인증절차를 우회할 수 있는 보안약점이다.

나. 안전한 코딩기법

XQuery 또는 XPath 쿼리에 사용되는 외부 입력데이터에 대하여 특수문자 및 쿼리 예약어를 필터링 하고 파라미터화(Parameterized)된 쿼리문을 지원하는 XQuery를 사용한다.

다. 코드예제

다음 예제는 Python에서 XML 데이터를 처리하기 위한 기본 모듈인 xml.etree.ElementTree를 이용하여 사용자 정보를 가져오는 예제이다. xml.etree.ElementTree 모듈은 제한적인 Xpath 기능을 제공하고 Xpath표현식을 매개변수화 해서 사용하는 방법을 제공하지 않는다.

안전하지 않은 코드의 예

```
1:  from lxml import etree
2:
3:  def parse_xml(request):
4:      user_name = request.POST.get('user_name', '')
5:
6:      parser = etree.XMLParser(resolve_entities=False)
7:      tree = etree.parse('user.xml', parser)
8:      root = tree.getroot()
9:
10:     # 검증되지 않은 외부 입력값 user_name를 사용하여
11:     # 안전하지 않은 질의문이 작성되에 query변수에 저장
12:     path = "/collection/users/user[@name='" + user_name + "']/home/text()"
13:     elmts = root.xpath(query)
14:     return render(request, 'parse_xml.html', {'xml_element':elmts})
```

Python 3.3 이후 보안상의 이유로 금지된 xml.etree.ElementTree 모듈 대신 lxml 라이브러리를 사용하고, 외부 입력값은 매개변수화 해서 사용한다.

안전한 코드의 예

```
1:  from lxml import etree
2:
3:  def parse_xml(request):
4:      user_name = request.POST.get('user_name', '')
5:
6:      parser = etree.XMLParser(resolve_entities=False)
7:      tree = etree.parse('user.xml', parser)
8:      root = tree.getroot()
9:
10:     query = '/collection/users/user[@name = $paramname]/home/text()'
11:     #외부 입력값을 paramname으로 매개변수화 해서 사용
12:     elmts = root.xpath(query, paramname=user_name)
13:     return render(request, 'parse_xml.html', {'xml_element':elmts})
```

라. 참고자료

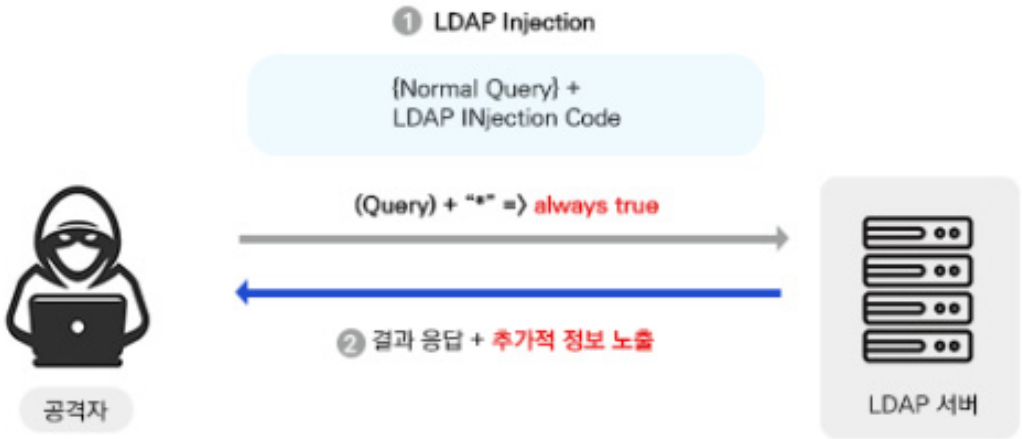
① CWE-643: Improper Neutralization of Data within XPath Expressions ('XPath Injection'), MITRE, https://cwe.mitre.org/data/definitions/643.html

② XPATH Injection, OWASP, https://owasp.org/www-community/attacks/XPATH_Injection

③ XML vulnerabilities, Python Software Foundation, https://docs.python.org/3/library/xml.html#xml-vulnerabilities

10. LDAP 삽입

가. 개요



공격자가 외부 입력을 통해서 의도하지 않은 LDAP(LightweightDirectoryAccessProtocol) 명령어를 수행할 수 있다. 즉, 웹 응용프로그램이 사용자가 제공한 입력을 올바르게 처리하지 못하면, 공격자가 LDAP 명령문의 구성을 바꿀 수 있다. 이로 인해 프로세스가 명령을 실행한 컴포넌트와 동일한 권한(Permission)을 가지고 동작하게 된다.

외부 입력값을 적절한 처리 없이 LDAP 쿼리문이나 결과의 일부로 사용하는 경우, LDAP 쿼리문이 실행될 때 공격자는 LDAP 쿼리문의 내용을 마음대로 변경할 수 있다.

Python에는 Python-ldap 및 ldap3라는 두 개의 라이브러리가 있다. ldap3가 python-ldap 보다 더 현대적인 라이브러리이다. ldap3 모듈은 Python 2.6부터 모든 Python 3 버전에 호환된다. ldap3에서는 좀 더 Python적인 방식으로 LDAP서버와 상호 작용할 수 있는 완전한 기능의 추상화 계층이 포함되어 있다.

python-ldap은 OpenLDAP에서 만든 Python2의 패키지로 Python3에서는 ldap3 라이브러리를 사용하는 것이 권장된다.

나. 안전한 코딩기법

다른 삽입 공격과 마찬가지로 LDAP 삽입에 대한 기본적인 방어 방법은 적절한 유효성 검사이다.

- 올바른 인코딩(Encoding) 함수를 사용하여 모든 변수 이스케이프(Escape)
- 화이트리스트 방식의 입력 값 유효성 검사
- 사용자 비밀번호와 같은 민감한 정보가 포함된 필드 인덱싱
- LDAP 바인딩 계정에 할당된 권한 최소화

다. 코드예제

사용자의 입력을 바로 LDAP 질의문에 사용하고 있으며 이 경우 권한 상승 등의 공격에 노출될 수 있다.

안전하지 않은 코드의 예

```
1:  from  ldap3 import  Connection, Server, ALL
2:  from django.shortcuts import render
3:
4:  def ldap_query(request):
5:      # 외부 입력값으로 LDAP 쿼리문의 인자에 검증 없이 사용하면
6:      # 안전하지 않다.
7:      search_keyword = request.POST.get('search_keyword','')
8:
9:      dn = server.config['bind_dn']
10:     password = server.config['password']
11:
12:     address = 'ldap.badSoruce.com'
13:     server = Server(address, get_info=ALL)
14:     conn = Connection(server, user=dn, password, auto_bind=True )
15:
16:     # 사용자 입력을 필터링 하지 않는 경우 공격자의 권한 상승으로
17:     # 이어질 수 있다
18:     search_str = '(&(objectclass=%s))' % search_keyword
19:
20:     conn.search('dc=company,dc=com', search_str, attributes=['sn', 'cn', 'address', 'mail', 'mobile', 'uid'])
21:     return render(request, '/ldap_query_response.html', {'ldap':conn.entries})
```

사용자의 입력 중 LDAP 질의문에 사용될 변수를 이스케이프 하여 질의문을 실행 시 공격에 노출되는 것을 예방할 수 있다.

안전한 코드의 예

```
1:  from  ldap3 import  Connection, Server, ALL
2:  from  ldap3.utils.conv import  escape_filter_chars
3:  from django.shortcuts import render
4:
5:  def ldap_query(request):
6:      search_keyword = request.POST.get('search_keyword','')
7:
8:      dn = server.config['bind_dn']
9:      password = server.config['password']
10:
11:     address = 'ldap.goodsource.com'
12:     server = Server(address, get_info=ALL)
13:     conn = Connection(server, dn, password, auto_bind=True )
14:
15:     # 사용자의 입력에 필터링을 적용하여 공격에 사용될 수 있는 문자를
16:     # 이스케이프하고 있다
17:     escpae_keyword = escape_filter_chars(search_keyword)
19:     search_str = '(&(objectclass=%s))' % escpae_keyword
20:
21:
22: conn.search('dc=company,dc=com', search_str, attributes=['sn', 'cn', 'address', 'mail', 'mobile', 'uid'])
        return render(request, '/ldap_query_response.html', {'ldap':conn.entries})
```

라. 참고자료

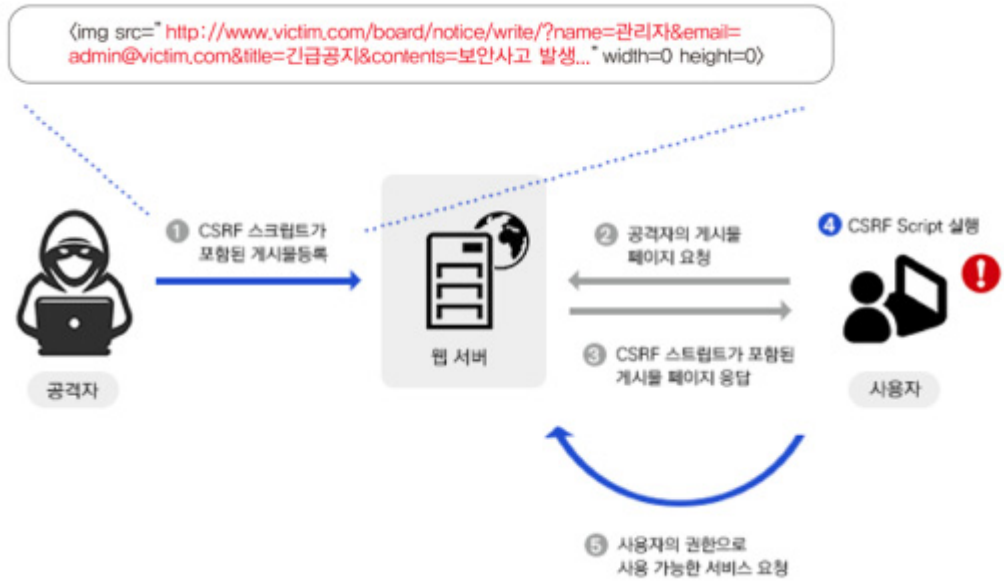
① CWE-90: Improper Neutralization of Special Elements used in an LDAP Query ('LDAP Injection'), MITRE, https://cwe.mitre.org/data/definitions/90.html

② LDAP Injection Prevention Cheat Sheet, OWASP, https://cheatsheetseries.owasp.org/cheatsheets/LDAP_Injection_Prevention_Cheat_ Sheet.html

③ LDAP filter handling, python-ldap project team https://www.python-ldap.org/en/python-ldap-3.3.0/reference/ldap-filter.html

11. 크로스사이트 요청 위조(CSRF)

가. 개요



특정 웹사이트에 대해서 사용자가 인지하지 못한 상황에서 사용자의 의도와는 무관하게 공격자가 의도한 행위(수정, 삭제, 등록 등)를 요청하게 하는 공격을 말한다. 웹 응용프로그램이 사용자로부터 받은 요청에 대해서 사용자가 의도한 대로 작성되고 전송된 것인지 확인하지 않는 경우 발생 가능하고 특히 해당 사용자가 관리자인 경우 사용자 권한관리, 게시물 삭제, 사용자 등록 등 관리자 권한으로만 수행 가능한 기능을 공격자의 의도대로 실행시킬 수 있게 된다.

공격자는 사용자가 인증한 세션이 특정 동작을 수행하여도 계속 유지되어 정상적인 요청과 비정상적인 요청을 구분하지 못하는 점을 악용한다.

파이썬에서 가장 많이 사용하고 있는 Django 프레임워크와 Flask 프레임워크에서는 각각 CSRF 토큰 기능을 지원하고 있으며, Django는 {% csrf token %} 태그를 이용하여 CSRF 토큰 기능 제공하고 Flask에서는 Flask-WTF 확장 라이브러리를 통해 {{form.csrf_token}} 태그를 이용하여 CSRF 토큰 기능을 제공하여 태그를 사용하는 경우 안전하게 사용할 수 있다.

나. 안전한 코딩기법

해당 요청이 정상적인 사용자의 정상적인 절차에 의한 요청인지를 구분하기 위해 세션별로 CSRF토큰을 생성하여 세션에 저장하고, 사용자가 작업페이지를 요청할 때마다 hidden값으로 클라이언트에게 토큰을 전달한 뒤, 해당 클라이언트의 데이터 처리 요청 시 전달되는 CSRF토큰값을 체크하여 요청의 유효성을 검사하도록 한다.

Django 프레임워크와 Flask 프레임워크는 미들웨어와 프레임워크에서 기본적으로 CSRF Token을 사용해서 CSRF 공격으로부터 보호하는 기능을 가지고 있다. 해당 기능을 사용하기 위해 form태그 내부에 csrf_token을 사용해야 한다.

다. 코드예제

가) Django 프레임워크 사용

Django프레임워크에서는 1.2 버전부터 CSRF 취약점을 방지기능을 기본으로 제공하고 있다. 미들웨어의 CSRF 옵션을 비활성하거나 템플릿에서 csrf_exempt decorator를 사용하는 경우, 크로스사이트 요청 위조 공격에 노출될 수 있다.

• Django 미들웨어 설정(settings.py) 사례

안전하지 않은 코드의 예

```
1:  MIDDLEWARE = [
2:      'django.contrib.sessions.middleware.SessionMiddleware',
3:      # MIDDLEWARE 목록에서 csrf 항목을 삭제 또는 주석처리 하면
4:      # Django 앱에서 csrf 유효성 검사가 전역적으로 제거 됨
5:      # 'django.middleware.csrf.CsrfViewMiddleware',
6:      'django.contrib.auth.middleware.AuthenticationMiddleware',
7:      'django.contrib.messages.middleware.MessageMiddleware',
8:      'django.middleware.locale.LocaleMiddleware',
9:      ......
10: ]
```

다음은 Django의 csrf 기능을 활성화하기 위한 안전한 미들웨어 설정 예제 이다.

미들웨어의 csrf 기능을 주석, 또는 삭제 처리 하지 않아야 한다. 템플릿 페이지에는 csrf_token을 form 태그 안에 명시해야 미들웨어에서 정상적으로 csrf 기능을 사용할 수 있다.

안전한 코드의 예

```
1:  MIDDLEWARE = [
2:      'django.contrib.sessions.middleware.SessionMiddleware',
3:      # MIDDLEWARE 목록에서 csrf 항목을 활성화 한다.
4:      'django.middleware.csrf.CsrfViewMiddleware',
5:      'django.contrib.auth.middleware.AuthenticationMiddleware',
6:      'django.contrib.messages.middleware.MessageMiddleware',
7:      'django.middleware.locale.LocaleMiddleware',
8:      ......
9:  ]
```

• Django 뷰 기능 설정(views.py) 사례

미들웨어에 csrf 검증 기능이 활성화 되어 있어도 View에서 csrf기능을 해제 하는 경우에는 해당 요청에 대해서 csrf 검증 기능을 사용하지 않게 된다.

다음은 Function-Based View에서 csrf 검증 기능을 비활성화 하는 예제이다.

안전하지 않은 코드의 예

```
1:  from django.shortcuts import render
2:  from django.views.decorators.csrf import csrf_exempt
3:
4:  # csrf.exempt 데코레이터로 미들웨어에서 보호되는 CSRF 기능을 해제한다.
5:  @csrf.exempt
6:  def pay_to_point(request):
7:      user_id = request.POST.get('user_id', '')
8:      pay = request.POST.get('pay', '')
9:      product_info = request.POST.get('product_info', '')
10:
11:     ret = pay(user_id, pay, product_info)
12:
13:     return render(request, '/view_wallet.html', {'wallet':ret})
```

Django는 기본적으로 CSRF 기능을 강제하고 있지만 몇몇 기능에 대해서 CSRF 기능을 해제 하여야 하는 경우는 미들웨어의 csrf 기능을 전역적으로 disable 하기 보다는 미들웨어의 csrf 기능은 활성화 하고 필요한 요청에 대해서만 csrf_exempt 데코레이터를 사용하여야 하고 이 경우에 크로스사이트 요청 위조의 위협에 노출될 수 있으므로 주의를 기울여야 한다.

안전한 코드의 예

```
1:  from django.shortcuts import render
2:  from django.template import RequestContext
3:
4:  # csrf_exempt 데코레이터를 삭제하거나 주석처리한다.
5:  # @csrf_exempt
6:  def pay_to_point(request):
7:      user_id = request.POST.get('user_id', '')
8:      pay = request.POST.get('pay', '')
9:      product_info = request.POST.get('product_info', '')
10:
11:     ret = pay(user_id, pay, product_info)
12:
13:     return render(request, '/view_wallet.html', {'wallet':ret})
```

• **Django 템플릿 설정 사례**

미들웨어에서 csrf 기능을 활성화 하여도 템플릿 페이지에 csrf 토큰을 명시하지 않을 경우 csrf 검증 기능을 사용할 수 없다.

안전하지 않은 코드의 예

```
1: 〈!--html page--〉
2: 〈form action="" method="POST"〉
3: 〈!-- form 태그 내부에 csrf_token 미적용--〉
4:     〈table〉
5:         {{form.as_table}}
6:     〈/table〉
7:     〈input type="submit"/〉
8: 〈/form〉
```

미들웨어에서 csrf 기능을 활성화한 후에 템플릿 페이지에서는 csrf_token 값을 명시하여야만 정상적인 csrf 검증 기능을 사용할 수 있다.

안전한 코드의 예

```
1: 〈!--html page--〉
2: 〈form action="" method="POST"〉
3:     {% csrf_token %}          〈!--csrf_token 사용-〉
4:     〈table〉
5:         {{form.as_table}}
6:     〈/table〉
7:     〈input type="submit"/〉
8: 〈/form〉
```

나) Flask 프레임워크 사용

• **Flask app 설정 사례**

Flask의 WTF 패키지를 사용하면 CSRF 보호 기법을 사용 할 수 있다.

아래 예제 코드는 CSRF 설정이 되지 않은 상태이다.

안전하지 않은 코드의 예

```
1: from flask import Flask
2:
3: app = Flask(__name__)
```

Flask 프레임워크를 사용하여 웹 애플리케이션을 구축하는 경우, CSRF를 방지하려면 Flask-WTF extension의 CSRFProtect를 사용해야 한다. app에 설정하고 HTML(템플릿) 페이지에는 CSRF토큰을 추가한다.

안전한 코드의 예

```
1:  from flask import Flask
2:  from flask_wtf.csrf import CSRFProtect
3:
4:  # CSRF 설정 추가
5:  csrf = CSRFProtect(app)
6:  app = Flask(__name__)
7:  app.config['SECRET_KEY'] = os.environ.get('SECRET_KEY')
8:  csrf.init_app(app)
```

• Flask 템플릿 설정 사례

위 코드처럼 함수에 CSRF 기능을 활성화 하여도 Html 파일에 csrf_token을 명시 하지 않을 경우 CSRF 검증 기능을 사용할 수 없다.

안전하지 않은 코드의 예

```
1:  〈form action="" method="POST"〉
2:  〈!-- form 태그 내부에 csrf_token 미적용--〉
3:      〈table〉
4:          {{as_table}}
5:      〈/table〉
6:      〈input type="submit"/〉
7:  〈/form〉
```

템플릿 페이지에도 csrf_token 값을 명시해줘야 정상적인 csrf 검증이 수행된다.

FlaskForm 사용 시에는 {{ form.csrf_token }}을 명시해야 하고 템플릿에 FlaskForm을 사용하지 않을 경우에는 form 태그 안에 hidden input 값으로 {{ csrf_token }} 값을 명시해야 한다.

안전한 코드의 예

```
1:  〈form action="" method="POST"〉
2:      〈!-- form 태그 내부에 csrf_token 적용--〉
3:      〈input type="hidden" name="csrf_token" value="{{ csrf_token }}" /〉
4:      〈table〉
5:          {{table}}
6:      〈/table〉
7:      〈input type="submit"/〉
8:  〈/form〉
```

라. 참고자료

① CWE-352: Cross-Site Request Forgery (CSRF), MITRE,
https://cwe.mitre.org/data/definitions/352.html

② Cross Site Request Forgery (CSRF), OWASP,
https://owasp.org/www-community/attacks/csrf

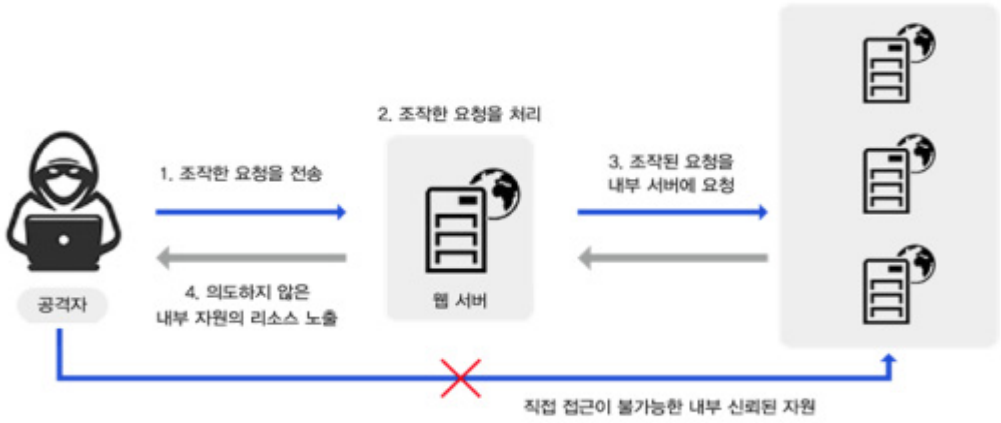
③ Cross-Site Request Forgery Prevention Cheat Sheet, OWASP
https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

④ Cross Site Request Forgery protection, Django Software Foundation
https://docs.djangoproject.com/en/3.2/ref/csrf/

⑤ CSRF Protection, WTForms
https://flask-wtf.readthedocs.io/en/0.15.x/csrf/

12. 서버사이드 요청 위조

가. 개요



적절한 검증절차를 거치지 않은 사용자 입력값을 서버간의 요청에 사용하여 악의적 인 행위가 발생할 수 있는 보안약점이다.

외부에 노출된 웹 서버에 취약한 애플리케이션이 존재하는 경우 공격자는 URL 또는 요청문을 위조하여 접근통제를 우회하는 방식으로 비정상적인 동작을 유도하거나 신뢰된 네트워크에 있는 데이터를 획득할 수 있다.

나. 안전한 코딩기법

식별할 수 있는 범위 내에서 사용자의 입력 값을 다른 시스템의 서비스 호출에 사용하는 경우, 사용자의 입력 값을 화이트리스트 방식으로 필터링한다.

사용자가 지정하는 무작위의 URL을 받아들여야 한다면 내부의 URL을 블랙리스트로 지정하여 필터링 한다. 또한 동일한 내부 네트워크에 있더라도 기기 인증, 접근권한을 확인하여 요청이 이루어질 수 있도록 한다.

다. 코드예제

〈참고 : 삽입 코드의 예〉

설명	삽입 코드의 예
내부망 중요 정보 획득	http://sample_site.com/connect?url=http://192.168.0.45/member/list.json
외부 접근 차단된 admin 페이지 접근	http://sample_site.com/connect?url=http://192.168.0.45/admin
도메인 체크를 우회하여 중요 정보 획득	http://sample_site.com/connect?url=http://sample_site.com:x@192.168.0.45/member/list.json
단축 URL을 이용한 Filter 우회	http://sample_site.com/connect?url=http://bit.ly/sdjk3kjhkl3
도메인을 사설IP로 설정 해 중요정보 획득	http://sample_site.com/connect?url=http://192.168.0.45/member/list.json
서버내 파일 열람	http://sample_site.com/connect?url=file:///etc/passwd

다음 예제는 안전하지 않은 코드 예제이다. 9라인의 사용자로부터 입력된 URL 주소를 검증 없이 사용하여 의도하지 않은 다른 서버의 자원에 접근 가능하게 된다.

안전하지 않은 코드의 예

```
1:  from django.shortcuts import render
2:  import requests
3:
4:  def call_third_party_api(request):
5:      addr = request.POST.get('address', '')
6:
7:      # 사용자가 입력한 주소를 검증하지 않고 HTTP 요청을 보내고 응답을
8:      # 사용자에게 반환
9:      result = requests.get(addr).text
10:     return render(request, '/result.html', {'result':result})
```

안전한 코드는 다음과 같이 사전에 정의된 서버 목록을 정의하고 매칭되는 URL만 사용할 수 있으므로 URL 값을 임의로 조작할 수 없다.

안전한 코드의 예

```
1:  from django.shortcuts import render
2:  import requests
3:
4:  # 도메인을 화이트리스트에 정의 할 경우 DNS rebinding 공격 등에
5:  # 노출될 위험이 있어 신뢰 할 수 있는 자원에 대한 IP를 사용하여
6:  # 검증하는 것이 좀더 안전하다.
7:  ALLOW_SERVER_LIST = [
8:      'https://127.0.0.1/latest/',
9:      'https://192.168.0.1/user_data',
10:     'https://192.168.0.100/v1/public'
11: ]
12:
13: def call_third_party_api(request):
14:     addr = request.POST.get('address', '')
15:
16:     # 사용자가 입력한 URL을 화이트리스트로 검증한 후 그 결과를 반환하여
17:     # 검증되지 않은 주소로 요청을 보내지 않도록 제한한다.
18:     if addr not in ALLOW_SERVER_LIST:
19:         return render(request, '/error.html', {error = '허용되지 않은 서버입니다.'})
20:
21:     result = requests.get(addr).text
22:     return render(request, '/result.html', {'result':result})
```

라. 참고자료

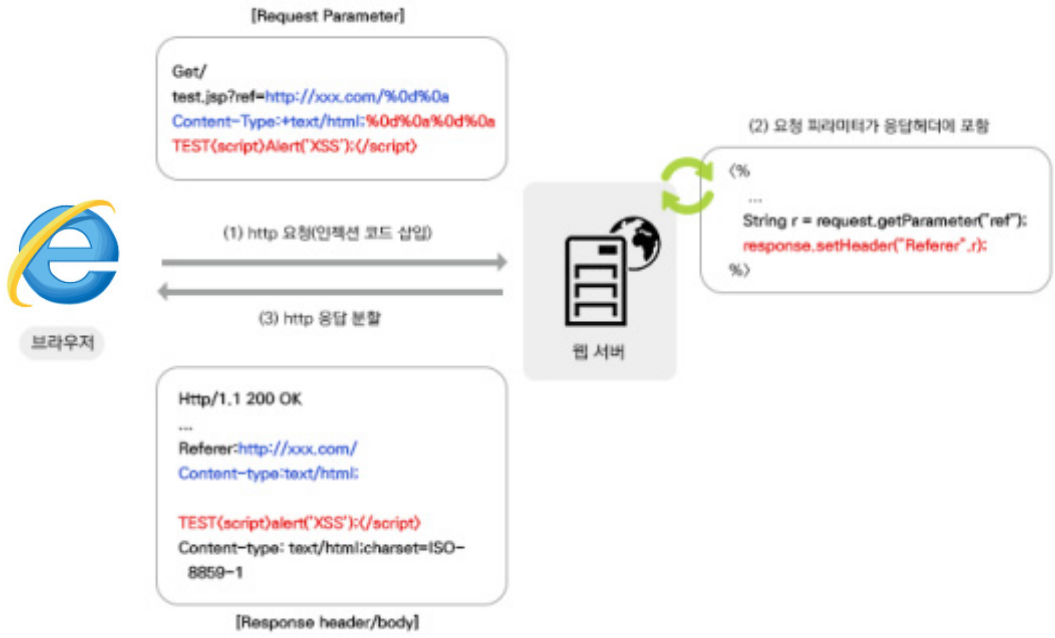
① CWE-918: Server-Side Request Forgery (SSRF), MITRE
https://cwe.mitre.org/data/definitions/918.html

② Server Side Request Forgery, OWASP
https://owasp.org/www-community/attacks/Server_Side_Request_Forgery

③ Server-Side Request Forgery Prevention Cheat Sheet, OWASP
https://cheatsheetseries.owasp.org/cheatsheets/Server_Side_Request_Forgery_Prevention_Cheat_Sheet.html

13. HTTP 응답분할

가. 개요



HTTP 요청에 들어 있는 파라미터(Parameter)가 HTTP 응답헤더에 포함되어 사용자에게 다시 전달될 때, 입력값에 CR(CarriageReturn)이나 LF(LineFeed)와 같은 개행문자가 존재하면 HTTP 응답이 2개 이상으로 분리될 수 있다. 이 경우 공격자는 개행문자를 이용하여 첫 번째 응답을 종료 시키고, 두 번째 응답에 악의적인 코드를 주입하여 XSS 및 캐시훼손(CachePoisoning) 공격 등을 수행할 수 있다.

Python 3.9.5+ 버전에서의 URLValidator에서 HTTP 응답분할 취약점이 보고되기도 했고 해당 라이브러리를 사용하는 Django버전에도 영향이 있다. HTTP 응답분할 공격으로부터 어플리케이션을 안전하게 지키려면 최신 버전의 라이브러리, 프레임 워크를 사용하고 외부 입력값에 대해서는 철저한 검증 작업을 수행해야 한다.

나. 안전한 코딩기법

요청 파라미터의 값을 HTTP 응답헤더(예를 들어, Set-Cookie 등)에 포함시킬 경우 CR(\r), LF(\n)와 같은 개행문자를 제거한다. 외부 입력값이 헤더, 쿠키, 로그등에 사용될 경우에는 항상 개행 문자를 검증하고 헤더에 사용되는 예약어 등은 화이트리스트로 제한 할 수 있으면 화이트리스트로 제한하여야 한다.

다. 코드예제

사용자 요청에 포함된 값을 필터링 및 검증 없이 응답에 사용하는 경우 개행문자로 인해 여러 개의 응답으로 분할되어 사용자에게 전달될 수 있다.

안전하지 않은 코드의 예

```
1:  from django.http import HttpResponse
2:
3:  def route(request):
4:      content_type = request.POST.get('contnet-type')
5:      # 외부 입력값을 검증 또는 필터링 하지 않고
6:      # 응답헤더의 값으로 포함시켜 회신한다.
7:      ......
8:      res = HttpResponse()
9:      res['Content-Type'] = content_type
10:     return res
```

응답 분할을 예방하기 위해 \r, \n과 같은 문자에 대해서 치환 또는 예외처리 하여 응답분할이 발생하지 않도록 예방하여야 한다.

안전한 코드의 예

```
1:  from django.http import HttpResponse
2:
3:  def route(request):
4:      content_type = request.POST.get('contnet-type')
5:
6:      # 응답헤더에 포함될 수 있는 외부 입력값에 대해 개행문자를 제거한다.
7:      content_type = content_type.replace('\r', '')
8:      content_type = content_type.replace('\n', '')
9:      ......
10:     res = HttpResponse()
11:     res['Content-Type'] = content_type
12:     return res
```

라. 참고자료

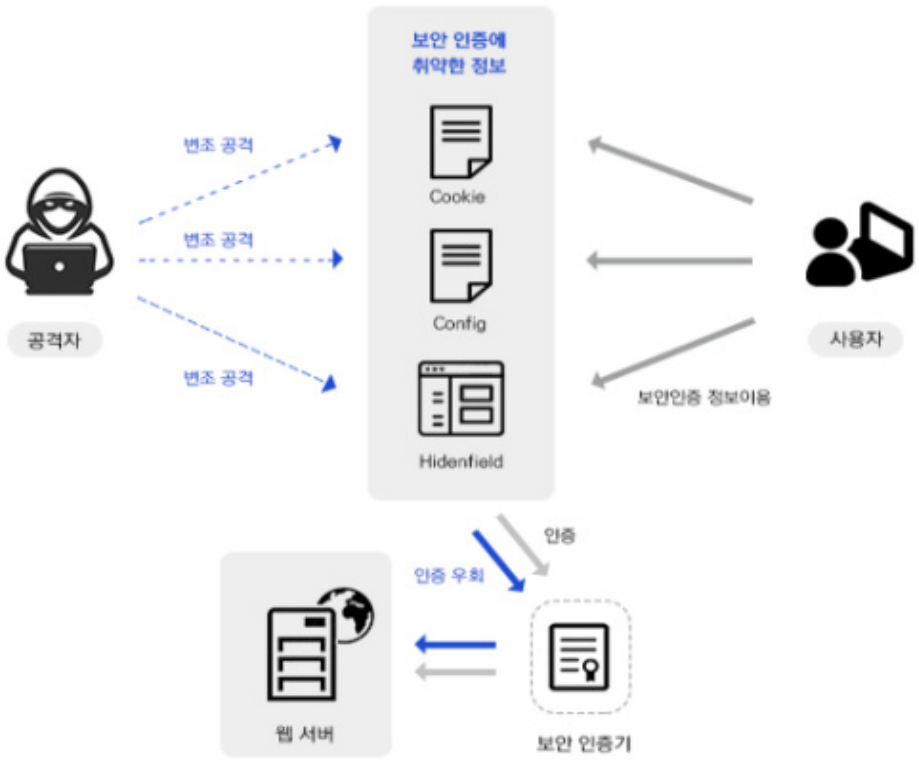
① CWE-113: Improper Neutralization of CRLF Sequences in HTTP Headers ('HTTP Response Splitting'), MITRE,
https://cwe.mitre.org/data/definitions/113.html

② HTTP Response Splitting, OWASP,
https://owasp.org/www-community/attacks/HTTP_Response_Splitting

③ Django security releases issued, Django Software Foundation,
https://www.djangoproject.com/weblog/2021/may/06/security-releases/

14. 보안기능 결정에 사용되는 부적절한 입력값

가. 개요



응용프로그램이 외부 입력값에 대한 신뢰를 전제로 보호메커니즘을 사용하는 경우 공격자가 입력값을 조작할 수 있다면 보호메커니즘을 우회할 수 있게 된다.

개발자들이 흔히 쿠키, 환경변수 또는 히든필드와 같은 입력값이 조작될 수 없다고 가정하지만 공격자는 다양한 방법을 통해 이러한 입력값들을 변경할 수 있고 조작된 내용은 탐지되지 않을 수 있다. 인증이나 인가와 같은 보안결정이 이런 입력값(쿠키, 환경변수, 히든필드 등)에 기반을 두어 수행되는 경우 공격자는 이런 입력값을 조작하여 응용프로그램의 보안을 우회할 수 있으므로 충분한 암호화, 무결성 체크를 수행하고 이와 같은 메커니즘이 없는 경우엔 외부사용자에 의한 입력값을 신뢰해서는 안 된다.

파이썬의 Djagno 프레임워크에서 세션을 관리하는 기능을 제공하고 있어 이 기능사용 시에는 세션쿠키의 만료 시점을 설정하여 사용할 수 있으며 DRF(Django Rest Framework)에서 제공하는 토큰 및 세션 기능을 사용하여 안전하게 구성할 수 있다.

나. 안전한 코딩기법

상태정보나 민감한 데이터 특히 사용자 세션정보와 같은 중요한 정보는 서버에 저장하고 보안확인 절차도 서버에서 실행한다. 보안설계관점에서 신뢰할 수 없는 입력 값이 응용프로그램 내부로 들어올 수 있는 지점과 보안결정에 사용되는 입력 값을 식별하고 제공되는 입력 값에 의존할 필요가 없는 구조로 변경할 수 있는지 검토한다.

다. 코드예제

다음은 안전하지 않은 코드로 쿠키에 저장된 권한 등급을 가져와 관리자인지 확인 후에 사용자의 패스워드를 초기화 하고 메일을 보내는 예제이다. 5라인에서 쿠키에서 등급을 가져와 9라인에서 관리자인지 확인하고 있다.

안전하지 않은 코드의 예

```
1:  from django.shortcuts import render
2:
3:  def init_password(request):
4:      # 쿠키에서 권한 정보를 가져옴
5:      roll = request.COOKIE['roll']
6:      request_id = request.POST.get('user_id', '')
7:      request_mail = request.POST.get('user_email','')
8:      # 쿠키에서 가져온 권한이 관리자인지 비교
9:      if roll == 'admin'
10:         # 사용자의 패스워드 초기화 및 메일 발송 처리
11:         password_init_and_sendmail(request_id, request_mail)
12:         return render(request, '/sucess.html')
13:     else:
14:         return render(request, '/failed.html')
```

중요기능을 수행하는 결정을 위한 데이터는 위변조 가능성이 높은 쿠키보다 세션에 저장하도록 한다.

안전한 코드의 예

```
1:  from django.shortcuts import render
2:
3:  def init_password(request):
4:      # 세션에서 권한 정보를 가져옴
5:      roll = request.session['roll']
6:      request_id = request.POST.get('user_id', '')
7:      request_mail = request.POST.get('user_email','')
8:      # 세션에서 가져온 권한이 관리자인지 비교
9:      if roll == 'admin'
10:         # 사용자의 패스워드 초기화 및 메일 발송 처리
11:         password_init_and_sendmail(request_id, request_mail)
12:         return render(request, '/sucess.html')
13:     else:
14:         return render(request, '/failed.html')
```

라. 참고자료

① CWE-807: Reliance on Untrusted Inputs in a Security Decision, MITRE, https://cwe.mitre.org/data/definitions/807.html

② How to use sessions, Django Software Foundation, https://docs.djangoproject.com/en/3.2/topics/http/sessions/

③ Flask Sessions, https://flask-session.readthedocs.io/en/latest/

15. 포맷 스트링 삽입

가. 개요

외부로부터 입력된 값을 검증하지 않고 입·출력 함수의 포맷 문자열로 그대로 사용하는 경우 발생할 수 있는 보안약점이다. 공격자는 포맷 문자열을 이용하여 취약한 프로세스를 공격하거나 메모리 내용을 읽거나 쓸 수 있다. 그 결과, 공격자는 취약한 프로세스의 권한을 취득하여 임의의 코드를 실행 할 수 있다.

Python에서는 문자열의 포맷팅 방법으로 “% formatting”, “str.format”, “f-string” 3가지 방법의 문자열 포맷팅 형식을 제공하고 있다. f-string 방법은 Python 3.6부터 도입된 포맷팅 방법이다. 공격자는 포맷 문자열을 이용하여 내부 정보를 문자열로 만들 수 있으며, 이를 반환하는 경우 내부의 중요 정보가 유출될 수 있다.

나. 안전한 코딩기법

포맷 문자열을 사용하는 함수를 사용할 때는 사용자 입력 값을 직접적으로 포맷 문자열로 사용하거나 포맷 문자열 생성에 포함시키지 않아야 한다. 사용자에게 입력받은 데이터를 포맷 문자열로 사용하고자 하는 경우에는 서식지정자를 포함하지 않거나 파이썬의 내장함수 또는 내장변수 등이 포함되지 않도록 해야 한다.

다. 코드예제

아래 예시에서는 외부에서 입력받은 문자열을 바로 포맷스트링으로 사용하고 있는데 이는 내부 정보가 외부로 노출될 수 있는 문제를 내포하고 있다.

공격자가 # {user.__init__.__globals__[AUTHENTICATE_KEY]} 형식의 문자열 입력 시 글로벌 변수에 접근하여 AUTHENTICATE_KEY의 값을 탈취 할 수 있다.

안전하지 않은 코드의 예

```
1:  from django.shortcuts import render
2:  AUTHENTICATE_KEY = 'Passw0rd'
3:
4:  def make_user_message(request):
5:      user_info = get_user_info(request.POST.get('user_id', ''))
6:
7:      format_string = request.POST.get('msg_format', '')
8:      # 내부의 민감한 정보가 외부로 노출될 수 있다.
9:      # 사용자가 입력한 문자열을 포맷 문자열로 사용하고 있어 안전하지 않다
10:     message = format_string.format(user=user_info)
11:
12:     return render(request, '/user_page', {'message':message})
```

외부에서 입력받은 문자열은 반드시 포맷지정자를 이용하여 바인딩하여 사용하여야 하며 직접적으로 포맷문자열로 사용해서는 안 된다.

안전한 코드의 예

```
1:  from django.shortcuts import render
2:  AUTHENTICATE_KEY = 'Passw0rd'
3:
4:  def make_user_message(request):
5:      user_info = get_user_info(request.POST.get('user_id', ''))
6:
7:      # 사용자가 입력한 문자열을 포맷 문자열로 사용하지 않아 안전하다
8:      message = 'user name is {}'.format(user_info.name)
9:
10:     return render(request, '/user_page', {'message':message})
```

라. 참고자료

① CWE-134: Use of Externally-Controlled Format String, MITRE, https://cwe.mitre.org/data/definitions/134.html

② Format string attack, OWASP, https://owasp.org/www-community/attacks/Format_string_attack

③ Python format, Python Software Foundation, https://docs.python.org/3/library/functions.html#format

④ Format String Syntax, Python Software Foundation, https://docs.python.org/3/library/string.html#format-string-syntax

제2절 보안기능

보안기능(인증, 접근제어, 기밀성, 암호화, 권한관리 등)을 부적절하게 구현 시 발생할 수 있는 보안약점으로 적절한 인증 없는 중요기능 허용, 부적절한 인가 등이 포함된다.

1. 적절한 인증 없는 중요 기능 허용

가. 개요

보안기능(인증, 접근제어, 기밀성, 암호화, 권한관리 등)을 부적절하게 구현 시 발생할 수 있는 보안약점으로 적절한 인증 없는 중요기능 허용, 부적절한 인가 등이 포함된다.

파이썬의 Django 프레임워크에서 django.contrib.auth 앱을 통하여 기본적인 인증 로그인 및 로그아웃 기능을 제공하고 있으며 DRF(Django REST Framework)에서는 토큰 및 세션 인증을 제공하고 있다.

나. 안전한 코딩기법

클라이언트의 보안검사를 우회하여 서버에 접근하지 못하도록 설계하고 중요한 정보가 있는 페이지는 재 인증을 적용한다. 또한 안전하다고 검증된 라이브러리나 프레임워크 (Django authentication system, Flask-Login 등)를 사용하는 것이 안전하다.

다. 코드예제

패스워드 수정 시 수정을 요청한 패스워드와 DB에 저장된 사용자 패스워드 일치 여부를 확인하지 않고 처리하고 있으며 패스워드의 재확인 절차도 생략되어 있다.

안전하지 않은 코드의 예

```
1:  from django.shortcuts import render
2:  from re import escape
3:  import hashlib
4:
5:  def change_password(request):
6:      new_pwd = request.POST.get('new_password','')
7:
8:      #로그인한 사용자정보
9:      user = '%s' % escape(request.session['userid'])
10:
11:     #현재 password와 일치여부를 확인하지 않고 수정함
12:     sha = hashlib.sha256(new_pwd.encode())
13:     update_password_from_db(user, sha.hexdigest())
14:
15:     return render(request, '/success.html')
```

DB에 저장된 사용자 패스워드와 변경 요청한 패스워드의 일치 여부를 확인하거나 변경 요청한 패스워드와 재확인 패스워드가 일치하는지 확인 후 DB의 패스워드를 수정하도록 하여 안전하게 적용할 수 있다.

안전한 코드의 예

```
1:  from django.shortcuts import render
2:  from re import escape
3:  import hashlib
4:
5:  # login_required decorator를 사용하여 login된 사용자만 접근하도록 처리
6:  @login_required
7:  def change_password(request):
8:      new_pwd = request.POST.get('new_password','')
9:      crnt_pwd = request.POST.get('current_password','')
10:
11:     # 세션에서 로그인한 사용자정보를 가져온다.
12:     user = '%s' % escape(request.session['userid'])
13:
14:     crnt_h = hashlib.sha256(crnt_pwd.encode())
15:     h_pwd = crnt_h.hexdigest()
16:
17:     # DB에서 기존 사용자의 Hash된 패스워드 가져오기
18:     old_pwd = get_password_from_db(user)
19:
20:     # 패스워드를 변경하기 전 사용자에 대해 재인증한다.
21:     if old_pwd == h_pwd:
22:         new_h = hashlib.sha256(new_pwd.encode())
23:         update_password_from_db(user, new_h.hexdigest())
24:         returnrender(request, '/success.html')
```

라. 참고자료

① CWE-306: Missing Authentication for Critical Function, MITRE,
https://cwe.mitre.org/data/definitions/306.html

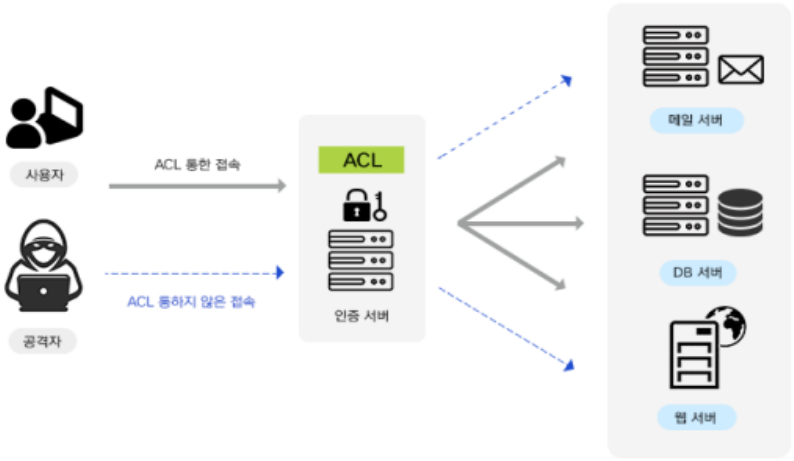
② Access Control, OWASP,
https://www.owasp.org/index.php/Access_Control_Cheat_Sheet

③ Using the Django authentication system, Django Software Foundation,
https://docs.djangoproject.com/en/3.2/topics/auth/default/

④ Flask-Security,
https://flask-login.readthedocs.io/en/latest/

2. 부적절한 인가

가. 개요



프로그램이 모든 가능한 실행 경로에 대해서 접근 제어를 검사하지 않거나 불완전하게 검사하는 경우, 공격자는 접근 가능한 실행경로를 통해 정보를 유출할 수 있다.

나. 안전한 코딩기법

응용프로그램이 제공하는 정보와 기능을 역할에 따라 배분함으로써 공격자에게 노출되는 공격노출면 (Attack Surface)을 최소화하고 사용자의 권한에 따른 ACL(Access Control List)을 관리한다.

다. 코드예제

사용자 입력 값에 따라 삭제작업을 수행하고 있으며, 사용자의 권한 확인을 위한 별도의 통제가 적용되지 않고 있다.

안전하지 않은 코드의 예

```
1:  from django.shortcuts import render
2:  from .model import Content
3:
4:  def delete_content(request):
5:      action = request.POST.get('action', '')
6:      content_id = request.POST.get('content_id', '')
7:      #작업 요청을 하는 사용자의 권한 확인 없이 delete를 수행
8:      if action is not None and action == "delete":
9:          Content.objects.filter(id=content_id).delete()
10:         return render(request, '/success.html')
11:     else:
12:         return render(request, '/error.html', {'error':'접근 권한이 없습니다.'})
```

세션에 저장된 사용자 정보를 통해 해당 사용자가 수행할 작업에 대한 권한이 있는지 확인한 뒤 권한이 있는 경우에만 수행하도록 해야 한다.

안전한 코드의 예

```
1:  from django.shortcuts import render
2:  from .model import Content
3:
4:  @login_required
5:   # 해당 기능을 수행할 권한이 있는지 확인
6:  @permission_required('content.delete', raise_exception=True)
7:  def delete_content(request):
8:      action = request.POST.get('action', '')
9:      content_id = request.POST.get('content_id', '')
10:
11:     if action is not None and action == "delete":
12:         Content.objects.filter(id=content_id).delete()
13:         return render(request, '/success.html')
14:     else:
15:         return render(request, '/error.html', {'error':'삭제 실패'})
```

라. 참고자료

① CWE-285: Improper Authorization, MITRE,
https://cwe.mitre.org/data/definitions/285.html

② Access Control, OWASP,
https://www.owasp.org/index.php/Access_Control_Cheat_Sheet

③ Using the Django authentication system, Django Software Foundation,
https://docs.djangoproject.com/en/3.2/topics/auth/default/

3. 중요한 자원에 대한 잘못된 권한 설정

가. 개요

응용프로그램이 중요한 보안관련 자원에 대하여 읽기 또는 수정하기 권한을 의도하지 않게 허가할 경우, 권한을 갖지 않은 사용자가 해당 자원을 사용하게 된다.

Python의 os.fchmod, os.chmod 등의 함수를 사용하여 파일 생성 및 읽기 모드에서 권한을 설정 할 수 있다.

나. 안전한 코딩기법

설정파일, 실행파일, 라이브러리 등은 SW 관리자에 의해서만 읽고 쓰기가 가능하도록 설정하고 설정 파일과 같이 중요한 자원을 사용하는 경우, 허가 받지 않은 사용자가 중요한 자원에 접근 가능 한지 검사한다.

다. 코드예제

다음 예제는 /root/system_config 파일에 대해서 모든 사용자가 읽기, 쓰기, 실행 권한을 가지게 된다.

안전하지 않은 코드의 예

```
1:  def write_file():
2:      #모든 사용자가 읽기, 쓰기, 실행 권한을 가지게 됨.
3:      os.chmod('/root/system_config', 0o777)
4:
5:      with open("/root/system_config",  'w') as f:
6:          f.write("your config is broken")
```

파일에 대해서는 최소권한을 할당해야 한다. 즉 해당 파일의 소유자에게만 읽기 권한만 부여하고 쓰기 권한이 필요한 경우에만 쓰기 권한을 부여한다.

안전한 코드의 예

```
1:  def write_file():
2:      #소유자 외에는 아무런 권한을 주지 않음.
3:      os.chmod('/root/system_config', 0o700)
4:
5:      with open("/root/system_config", "w') as f:
6:          f.write("your config is broken")
```

라. 참고자료

① CWE-732: Incorrect Permission Assignment for Critical Resource, MITRE, https://cwe.mitre.org/data/definitions/732.html

② OS - Miscellaneous operating system interfaces, Python Software Foundation, https://docs.python.org/3/library/os.html

4. 취약한 암호화 알고리즘 사용

가. 개요

SW 개발자들은 환경설정 파일에 저장된 패스워드를 보호하기 위하여 간단한 인코딩 함수를 이용 하여 패스워드를 감추는 방법을 사용하기도 한다. 그렇지만 base64와 같은 지나치게 간단한 인코딩 함수로는 패스워드를 제대로 보호할 수 없다.

정보보호 측면에서 취약하거나 위험한 암호화 알고리즘을 사용해서는 안 된다. 표준화되지 않은 암호화 알고리즘을 사용하는 것은 공격자가 알고리즘을 분석하여 무력화시킬 수 있는 가능성을 높일 수도 있다. 몇몇 오래된 암호화 알고리즘의 경우는 컴퓨터의 성능이 향상됨에 따라 취약해지기도 해서, 예전에는 해독하는데 몇 십 억년이 걸릴 것이라고 예상되던 알고리즘이 며칠이나 몇 시간 내에 해독되기도 한다. RC2 (ARC2), RC4 (ARC4), RC5, RC6, MD4, MD5, SHA1, DES 알고리즘이 여기에 해당된다.

Python 2에서는 PyCrypto를 사용하였지만 Python 3에서는 PyCryto를 개선한 PyCryptodomes를 사용하여야 한다.

나. 안전한 코딩기법

자신만의 암호화 알고리즘을 개발하는 것은 위험하며, 학계 및 업계에서 이미 검증된 표준화된 알고리즘을 사용한다. 기존에 취약하다고 알려진 DES, RC5등의 암호알고리즘을 대신하여, 3DES, AES, SEED 등의 안전한 암호알고리즘으로 대체하여 사용한다. 또한, 업무관련 내용, 개인정보 등에 대한 암호 알고리즘 적용 시, 암호모듈시험기에서 안전성을 확인한 검증필 암호모듈을 사용해야 한다.

〈 암호알고리즘 검증기준 ver3.0 (암호모듈시험기관) 〉

분류		암호 알고리즘
최소 안전성 수준		• 112비트
블록암호 (운영모드)	ARIA	• 운영모드 – 기밀성(ECB, CBC, CFB, OFB, CTR) – 기밀성/인증(CCM, GCM)
	SEED	• 운영모드 – 기밀성(ECB, CBC, CFB, OFB, CTR) – 기밀성/인증(CCM, GCM)
	LEA	• 운영모드 – 기밀성(ECB, CBC, CFB, OFB, CTR) – 기밀성/인증(CCM, GCM)
	HIGHT	• 운영모드 – 기밀성(ECB, CBC, CFB, OFB, CTR)
해시함수	SHA-2	• SHA-224/256/384/512
	LSH	• LSH-224/256/384/512/512-224/512-256
	SHA-3	• SHA-3-224/256/384/512
메시지 인증	해시함수 기반	• HMAC
	블록암호 기반	• CMAC, GMAC
난수발생기	해시함수 기반	• Hash_DRBG, HMAC_DRBG
	블록암호 기반	• CTR_DRBG
공개키 암호	RSAES	• 공개키 길이 : 2048, 3072 • 해시함수 : SHA-224, SHA-256
전자서명	RSA-PSS	• 공개키 길이 : 2048, 3072 • 해시함수 : SHA-224, SHA-256
	KCDSA	• (공개키 길이, 개인키 길이) : (2048, 224), (2048, 256) • 해시함수 : SHA-224, SHA-256
	EC-KCDSA	• p-224, p-256, B-233, B-283, K-233, K-283 • 해시함수 : SHA-224, SHA-256
	ECDSA	• p-224, p-256, B-233, B-283, K-233, K-283 • 해시함수 : SHA-224, SHA-256
키 설정	DH	• (공개키 길이, 개인키 길이) : (2048, 224), (2048, 256)
	ECDH	• p-224, p-256, B-233, B-283, K-233, K-283
키 유도	KBKDF	• HMAC, CMAC
	PBKDF	• HMAC

다. 코드예제

다음 예제는 취약한 DES 알고리즘으로 암호화하고 있다. DES 이외에 3DES, Blowfish, ARC2, ARC4 등의 취약한 알고리즘을 사용하지 않는다.

안전하지 않은 코드의 예

```
1:  import base64
2:  from Crypto.Cipher import DES
3:
4:  def get_enc_text(plain_text, key):
5:      # 취약함 암호와 알고리즘인 DES를 사용하여 안전하지 않음
6:      cipher_des = DES.new(key, DES.MODE_ECB)
7:      encrypted_data = base64.b64encode(cipher_des.encrypt(plain_text))
8:
9:      return encrypted_data.decode('ASCII')
```

pycrypto 라이브러리는 python 3.x버전 환경에서 사용 시 동작을 하지 않는 경우가 발생하며, 더 이상 유지관리 되지 않으므로 pycrypto를 개선한 pycryptodome를 사용한다. 취약한 DES 알고리즘 대신 안전한 AES 암호화 알고리즘을 사용한다.

블록 암호화에서 운영모드를 ECB(Electronic Code Block) 모드를 사용할 경우 한 개의 블록만 해독되면 나머지 블록도 해독이 되는 단점이 있다. CBC(Cipher Block Chaining) 모드는 평문의 각 블록이 XOR연산을 통해 이전 암호문과 연산이 되기 때문에 같은 평문이라도 암호문이 서로 다르다. 이러한 특성으로 보안성이 ECB 모드보다 높다.

안전한 코드의 예

```
1:  import base64
2:  from Crypto.Cipher import AES
3:
4:  def get_enc_text(plain_text, key, iv):
5:      # 안전한 알고리즘인 AES 를 사용하여 안전함.
6:      cipher_aes = AES.new(key, AES.MODE_CBC, iv)
7:      encrypted_data = base64.b64encode(cipher_aes.encrypt(plain_text))
8:
9:      return encrypted_data.decode('ASCII')
```

다음 예제는 취약한 MD5 해쉬함수를 사용하고 있다.

안전하지 않은 코드의 예

```
1:  import hashlib
2:
3:  def make_md5(plain_text):
4:      # 취약한 md5 해쉬함수 사용
5:      hash_text = hashlib.md5(plain_text.encode('utf-8')).hexdigest()
6:
7:      return hash_text
```

아래 코드처럼 안전하다고 알려진 sha-256 해쉬함수 등을 적용해야 한다.

안전한 코드의 예

```
1:  import hashlib
2:
3:  def make_sha256(plain_text):
4:      # 안전한 sha-256 해쉬함수 사용
5:      hash_text = hashlib.sha256(plain_text.encode('utf-8')).hexdigest()
6:
7:      return hash_text
```

라. 참고자료

① CWE-327: Use of a Broken or Risky Cryptographic Algorithm, MITRE, https://cwe.mitre.org/data/definitions/327.html

② Welcome to pyca/cryptography, Cryptography, https://cryptography.io/en/latest/

③ Welcome to PyCryptodome's documentation, PyCryptodome, https://www.pycryptodome.org/en/latest/

④ Cryptographic Services, Python Software Foundation, https://docs.python.org/3/library/crypto.html

5. 암호화되지 않은 중요정보

가. 개요

많은 응용프로그램은 메모리나 디스크에서 중요한 정보(개인정보, 인증정보, 금융정보 등)를 처리한다. 이러한 중요정보가 제대로 보호되지 않을 경우, 보안이나 데이터의 무결성을 잃을 수 있다. 특히 사용자 또는 시스템의 중요정보가 포함된 데이터를 평문으로 송·수신 또는 저장할 때 인가되지 않은 사용자에게 민감한 정보가 노출될 수 있다.

나. 안전한 코딩기법

개인정보(주민등록번호, 여권번호 등), 금융정보(카드번호, 계좌번호 등), 패스워드 등 중요정보를 저장하거나 통신채널로 전송할 때는 반드시 암호화 과정을 거쳐야 하며 중요정보를 읽거나 쓸 경우에 권한인증 등을 통해 적합한 사용자가 중요정보에 접근하도록 해야 한다.

필요한 경우 SSL 또는 HTTPS 등과 같은 보안 채널을 사용해야 하며, HTTPS와 같은 보안 채널을 사용하거나 브라우저 쿠키에 중요 데이터를 저장하는 경우, 쿠키객체에 보안속성을 설정하여(Ex. secure = True) 중요정보의 노출을 방지한다.

다. 코드예제

• 중요정보 평문저장

아래 예제는 사용자로부터 전달받은 비밀번호 암호화를 누락한 경우이다.

안전하지 않은 코드의 예

```
1:  def update_pass(dbconn, password, user_id):
2:      curs = dbconn.curs()
3:      # 암호화되지 않은 비밀번호를 DB에 저장하는 경우
4:      curs.execute('UPDATE USERS SET PASSWORD=%s WHERE USER_ID=%s', password, user_id)
```

아래는 해쉬 알고리즘을 이용하여 단방향 암호화 이후에 비밀번호를 저장하고 있다.

안전한 코드의 예

```
1:  from Crypto.Hash import SHA256
2:
3:  def update_pass(dbconn, password, user_id, salt):
4:      # 단방향 암호화를 이용하여 비밀번호를 암호화
5:      hash_obj = SHA256.new()
6:      hash_obj.update(bytes(password + salt, 'utf-8'))
7:      hash_pwd = hash_obj.hexdigest()
8:      curs = dbconn.curs()
9:      curs.execute('UPDATE USERS SET PASSWORD=%s WHERE USER_ID=%s', (hash_pwd,
    user_id))
```

• 중요정보 평문전송

아래 예제는 인자값으로 전달 받은 패스워드를 검증 없이 네트워크를 통해 전송하고 있다. 전달 받은 패스워드가 암호화가 되어 있지 않을 경우 패킷 스니핑을 통하여 패스워드가 노출될 수 있다.

안전하지 않은 코드의 예

```
1:  import socket
2:
3:  HOST = '127.0.0.1'
4:  PORT = 65434
5:
6:  def send_password(password):
7:      .....
8:      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
9:          s.connect((HOST, PORT))
10:         # 패스워드를 암호화 하지 않고 전송하여 안전하지 않다.
11:         s.sendall(password.encode('utf-8'))
12:         data = s.recv(1024)
13:         .....
```

아래는 네트워크를 통해 전달되는 패스워드가 노출되지 않도록 암호화하여 전송하고 있다.

안전한 코드의 예

```
1:  import socket
2:  import os
3:  from Crypto.Cipher import AES
4:
5:  HOST = '127.0.0.1'
6:  PORT = 65434
7:
8:  def send_password(password):
9:      ......
10:     block_key = os.environ.get('BLOCK_KEY')
11:
12:     aes = AEScipher(block_key)
13:     # 패스워드등 중요정보는  암호화하여 전송하는게 안전하다.
14:     enc_passowrd = aes.encrypt(passowrd)
15:     with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
16:         s.connect((HOST, PORT))
17:         s.sendall(enc_passowrd.encode('utf-8'))
18:         data = s.recv(1024)
19:         .......
20:
21: class AEScipher:
22:     BS = AES.block_size
23:
24:     def __init__(self, s_key):
25:         self.s_key = s_key
26:
27:     def pad(m):
28:         return m + bytes([BS - len(m) % BS] * (BS - len(m) % BS))
29:
30:     def encrypt(self, plain):
31:         plain = pad(plain.encode())
32:         iv = os.environ.get('IV_KEY')
33:         cipher = AES.new(self.s_key, AES.MODE_CBC, iv)
34:         return (iv + cipher.encrypt(plain))
35:         ......
```

라. 참고자료

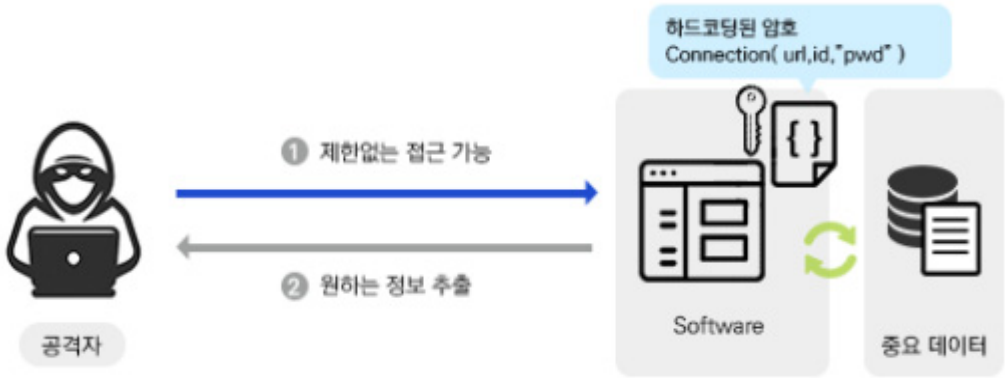
① CWE-312: Cleartext Storage of Sensitive Information, MITRE, https://cwe.mitre.org/data/definitions/312.html

② CWE-319: Cleartext Transmission of Sensitive Information, MITRE, https://cwe.mitre.org/data/definitions/319.html

③ Password Plaintext Storage, OWASP, https://owasp.org/www-community/vulnerabilities/Password_Plaintext_Storage

6. 하드코드된 중요정보

가. 개요



프로그램 코드 내부에 하드코드된 패스워드를 포함하고, 이를 이용하여 내부 인증에 사용하거나 외부 컴포넌트와 통신을 하는 경우 관리자의 정보가 노출 될 수 있어 위험하다. 또한 하드코드된 암호화 키를 사용하여 암호화를 수행하면 암호화된 정보가 유출될 가능성이 높아진다. 암호키의 해쉬를 계산하여 저장하더라도 역계산이 가능하여 적어도 무차별 공격(Brute-Force)공격에는 취약할 수 있다.

나. 안전한 코딩기법

패스워드는 암호화 하여 별도의 파일에 저장하여 사용한다. 또한 중요정보를 암호화하면, 상수가 아닌 암호화 키를 사용하도록 하며 암호화 되었더라도 소스코드 내부에 상수 형태의 암호화 키를 저장해서 사용하지 않도록 한다.

다. 코드예제

소스코드에 패스워드 또는 암호화 키 같은 중요 정보를 하드코딩 하는 경우, 중요정보가 노출될 수 있어 위험하다.

안전하지 않은 코드의 예

```
1:  import pymysql
2:  def query_execute(query):
3:      # usre, passwd가 소스코드에 평문으로 하드코딩되어 있음
4:      dbconn = pymysql.connect(host='127.0.0.1', port='1234', user='root', passwd='1234',
    db='mydb', charset='utf8')
5:      curs = dbconn.cursor()
6:      curs.execute(query)
7:      dbconn.commit()
8:      dbconn.close()
```

패스워드와 같은 중요정보는 안전한 암호화 방식으로 암호화하여 별도의 분리된 공간(파일)에 저장해야 하며, 암호화된 중요정보를 사용하기 위해서는 복호화 과정을 거쳐야 한다.

안전한 코드의 예

```
1:  import pymysql
2:  import json
3:
4:  def query_execute(query, config_path):
5:
6:      with open(config_path, 'r') as config:
7:          # 설정파일에서 user, passwd를 가져와 사용
8:          dbconf = json.load(fp=config)
9:        #암호화되어 있는 블록암호화 키를 복호화 해서 가져오는
10:         #사용자 정의 함수
11:         blockKey = get_decrypt_key('blockKey')
12:         # 설정파일에 암호화되어 있는 값을 가져와 복호화한 후에 사용
13:         dbUser = decrypt(blockKey, dbconf['user'])
14:         dbPasswd = decrypt(blockKey, dbconf['passwd'])
15:
16:         dbconn = pymysql.connect(host=dbconf['host'], port=dbconf['port'], user=dbUser, pass
    wd=dbPasswd, db=dbconf['db_name'], charset='utf8')
17:         curs = dbconn.cursor()
18:         curs.execute(query)
19:         dbconn.commit()
20:         dbconn.close()
```

라. 참고자료

① CWE-259: Use of Hard-coded Password, MITRE,
https://cwe.mitre.org/data/definitions/259.html

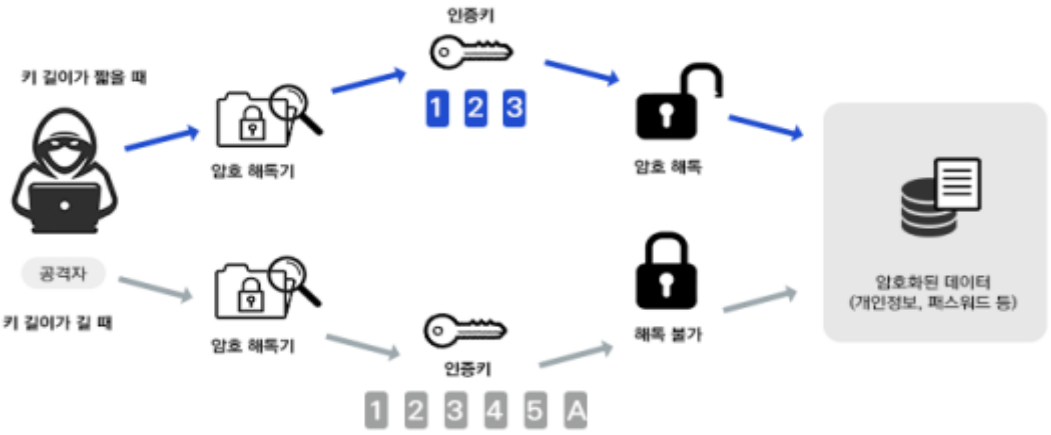
② CWE-321: Use of Hard-coded Cryptographic Key, MITRE,
https://cwe.mitre.org/data/definitions/321.html

③ Use of hard-coded password, OWASP,
https://owasp.org/www-community/vulnerabilities/Use_of_hard-coded_password

④ Password Management Hardcoded Password, OWASP,
https://owasp.org/www-community/vulnerabilities/Password_Management_Hardcoded_Password

7. 충분하지 않은 키 길이 사용

가. 개요



길이가 짧은 키를 사용하는 것은 암호화 알고리즘을 취약하게 만들 수 있다. 키는 암호화 및 복호화에 사용되는데, 검증된 암호화 알고리즘을 사용하더라도 키 길이가 충분히 길지 않으면 짧은 시간 안에 키를 찾아낼 수 있고 이를 이용해 공격자가 암호화된 데이터나 패스워드를 복호화 할 수 있게 된다.

암호 알고리즘 및 키 길이 선택 시, 암호 알고리즘의 안전성 유지기간과 보안강도별 암호 알고리즘 키 길이 비교표를 기반으로 암호 알고리즘 및 키 길이를 선택하여야 한다.

〈보안강도별 암호 알고리즘 비교표 – 암호알고리즘 및 키 길이 이용 안내서 (KISA)〉

보안강도	대칭키 암호 알고리즘 (보안강도)	해시함수 (보안강도)	공개키 암호 알고리즘				암호 알고리즘 안전성 유지기간 (년도)
			인수분해 (비트)	이산대수		타원곡선 암호(비트)	
				공개키(비트)	개인키(비트)		
112 비트	112	112	2048	2048	224	224	2011년에서 2030년까지
128 비트	128	128	3072	3072	256	256	2030년 이후
192 비트	192	192	7680	7680	384	384	
256 비트	256	256	15360	15360	512	512	

나. 안전한 코딩기법

RSA 알고리즘은 적어도 2,048 비트 이상의 길이를 가진 키와 함께 사용해야 하고, 대칭 암호화 알고리즘(Symmetric Encryption Algorithm)의 경우에는 적어도 128비트 이상의 키를 사용한다.

다. 코드예제

보안성이 강한 RSA 알고리즘을 사용하는 경우에도 키 사이즈를 작게 설정하면 프로그램의 보안약점이 발생할 수 있다.

안전하지 않은 코드의 예

```
1:  from Crypto.PublicKey import  RSA, DSA, ECC
2:  from tinyec import registry
3:  import secrets
4:
5:  def make_rsa_key_pair():
6:      # RSA키 길이를 1024 비트로 설정하는 경우 안전하지 않음
7:      private_key = RSA.generate(1024)
8:      public_key = private_key.publickey()
9:
10: def make_ecc():
11:     # 2015년부터 ECC 키 길이를 256 비트 이상으로 제안하고 있음.
12:     ecc_curve = registry.get_curve('secp192r1')
13:     private_key = secrets.randbelow(ecc_curve.field.n)
14:     public_key = private_key * ecc_curve.g
```

RSA, DSA의 경우 키의 길이는 적어도 2048 비트 ECC의 경우 256 비트 이상으로 설정한다.

아래 예제 중 ECC 키 생성은 tinyec 모듈을 사용하여 ECC 키를 생성한 예제이다.

안전한 코드의 예

```
1:  from Crypto.PublicKey import  RSA, DSA, ECC
2:  from tinyec import registry
3:  import secrets
4:
5:  def make_rsa_key_pair():
6:      # RSA키 길이를 2048 비트 이상으로 길게 설정
7:      private_key = RSA.generate(2048)
8:      public_key = private_key.publickey()
9:
10: def make_ecc():
11:     # ECC 키 길이를 256 비트 이상으로 설정.
12:     ecc_curve = registry.get_curve('secp256r1')
13:     private_key = secrets.randbelow(ecc_curve.field.n)
14:     public_key = private_key * ecc_curve.g
```

라. 참고자료

① CWE-326: Inadequate Encryption Strength, MITRE,
https://cwe.mitre.org/data/definitions/326.html

② FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION (FIPS PUB 186-4), NIST
https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf

③ PyCryptodome-RSA,
https://pycryptodome.readthedocs.io/en/latest/src/public_key/rsa.html

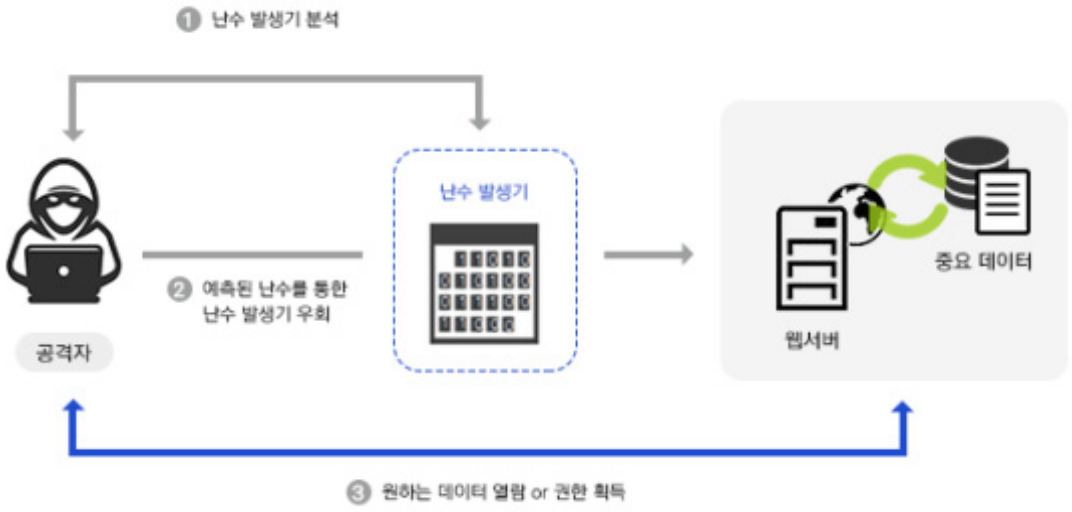
④ 암호 알고리즘 및 키 길이 이용 안내서, KISA,
https://www.kisa.or.kr/jsp/common/downloadAction.jsp?bno=259&dno=82&fseq=1

⑤ DSA, Pycryptodome,
https://pycryptodome.readthedocs.io/en/latest/src/public_key/dsa.html

⑥ ECC, Pycryptodome,
https://pycryptodome.readthedocs.io/en/latest/src/public_key/ecc.html

8. 적절하지 않은 난수 값 사용

가. 개요



예측 가능한 난수를 사용하는 것은 시스템에 보안약점을 유발한다. 예측 불가능한 숫자가 필요한 상황에서 예측 가능한 난수를 사용한다면, 공격자는 SW에서 생성되는 다음 숫자를 예상하여 시스템을 공격하는 것이 가능하다.

나. 안전한 코딩기법

난수 발생기에서 시드(Seed)를 사용하는 경우에는 고정된 값을 사용하지 않고 예측하기 어려운 방법으로 생성된 값을 사용한다.

python에서 random 모듈은 주로 보안목적이 아닌 게임, 퀴즈 및 시뮬레이션을 위해 설계되었다. 세션 ID, 암호화키 등 보안결정을 위한 값을 생성하고 보안결정을 수행하는 경우에는 암호화 목적으로 설계된 secrets 모듈을 사용한다.

secrets 모듈은 python 3.6 이상에서만 사용할 수 있으며 암호, 계정 인증, 보안 토큰과 같은 데이터를 관리하는데 적합한 강력한 난수를 생성하는 데 사용된다.

python 3.6 이하에서는 os.urandom(), random.SystemRandom 클래스를 사용하는 것이 안전하다.

다. 코드예제

random 라이브러리 사용시에는 반드시 유추하기 어려운 seed 값을 이용하여 난수를 생성하여야 하며 이렇게 생성된 난수라 하더라도 강도가 낮기 때문에 보안결정을 위한 난수 이용시에는 안전하지 않다. 아래는 안전하지 않은 코드 예제로 고정된 seed 값을 사용하였고 보안이나 암호를 목적으로 사용하기에 random 모듈은 안전하지 않다.

안전하지 않은 코드의 예

```
1:  import random
2:
3:  def get_otp_number():
4:      random_str = ''
5:      # 시스템 현재 시간 값을 시드로 사용, 보안결정을 위한
6:      # 난수로는 안전하지 않다.
7:      for i in range(6):
8:          random_str += str(random.randrange(10))
9:
10:     return random_str
```

다음 예제 코드는 secrets 라이브러리를 사용하여 6자리의 난수값을 생성하는 안전한 예제이다.

안전한 코드의 예

```
1:  import secrets
2:
3:  def get_otp_number():
4:      random_str = ''
5:
6:      # 보안기능에 적합한 난수 생성용 secrets 라이브러리 사용
7:      for i in range(6):
8:          random_str += str(secrets.randbelow(10))
9:
10:     return random_str
```

다음은 세션 토큰값을 생성하는 예제로 random 라이브러리를 사용하여 안전하지 않은 예제이다.

안전하지 않은 코드의 예

```
1:  import random
2:  import string
3:
4:  def generate_session_key():
5:      RANDOM_STRING_CHARS = string.ascii_letters+string.digits
6:      # random 라이브러리는 보안기능에 사용하면 위험하다
7:      return "".join(random.choice(RANDOM_STRING_CHARS) for i in range(32))
```

비밀번호나 인증정보 및 보안토큰 생성에 사용하는 경우 좀더 안전한 secrets 라이브러리를 사용하여 생성된 난수를 이용하여야 한다.

안전한 코드의 예

```
1:  import secrets
2:  import string
3:
4:  def generate_session_key():
5:      RANDOM_STRING_CHARS = string.ascii_letters+string.digits
6:      # 보안기능과 관련된 난수는 secrets 라이브러리를 사용해야 안전하다.
7:      return "".join(secrets.choice(RANDOM_STRING_CHARS) for i in range(32))
```

라. 참고자료

① CWE-330: Use of Insufficiently Random Values, MITRE,
https://cwe.mitre.org/data/definitions/330.html

② Insecure Randomness, OWASP,
https://owasp.org/www-community/vulnerabilities/Insecure_Randomness

③ Generate pseudo-random numbers, Python Software Foundation,
https://docs.python.org/3/library/random.html

④ Generate secure random numbers for managing secrets, Python Software Foundation,
https://docs.python.org/3/library/secrets.html

9. 취약한 비밀번호 허용

가. 개요

사용자에게 강한 패스워드 조합규칙을 요구하지 않으면, 사용자 계정이 취약하게 된다. 안전한 패스워드를 생성하기 위해서는 「패스워드 선택 및 이용 안내서」의 패스워드 설정규칙을 적용해야 한다.

나. 안전한 코딩기법

패스워드 생성 시 강한 조건 검증을 수행한다. 비밀번호(패스워드)는 숫자와 영문자, 특수문자 등을 혼합하여 사용하고, 주기적으로 변경하여 사용하도록 해야 한다.

다. 코드예제

사용자가 입력한 패스워드에 대한 복잡도 검증 없이 가입 승인 처리를 수행하고 있다.

안전하지 않은 코드의 예

```
1:  from flask import request, redirect
2:  from Models import User
3:  from Models import db
4:
5:  @app.route('/register', methods=['POST'])
6:  def register():
7:      userid = request.form.get('userid')
8:      password = request.form.get('password')
9:      confirm_password = request.form.get('confirm_password')
10:
11:     if password != confirm_password:
12:         return make_response("비밀번호가 일치하지 않습니다", 200)
13:     else:
14:         usertable=User()
15:         usertable.userid = userid
16:         usertable.password = password
17:         # 패스워드 생성 규칙을 확인하지 않고 회원 가입
18:         db.session.add(usertable)
19:         db.session.commit()
20:         return make_response("회원가입 성공", 200)
```

사용자 계정을 보호하기 위해 가입 시, 패스워드 복잡도와 길이를 검증 후 가입 승인처리를 수행한다.

안전한 코드의 예

```
from flask import request, redirect
from Models import User
from Models import db
import re

@app.route('/register', methods=['POST'])
def register():
    userid = request.form.get('userid')
    password = request.form.get('password')
    confirm_password = request.form.get('confirm_password')

    if password != confirm_password:
        return make_response("비밀번호가 일치하지 않습니다.", 200)

    if not check_password(password):
        return make_response("비밀번호 조합규칙에 맞지 않습니다.", 200)
    else:
        usertable=User()
        usertable.userid = userid
        usertable.password = password

        db.session.add(usertable)
        db.session.commit()
        return make_response("회원가입 성공", 200)
def check_password(password):
    # 2종 이상 문자로 구성된 8자리 이상 비밀번호 검사 정규식
    PT1 = re.compile('^(?=.*[A-Z])(?=.*[a-z])[A-Za-z\d!@#$%^&*]{8,}$')
    PT2 = re.compile('^(?=.*[A-Z])(?=.*\d)[A-Za-z\d!@#$%^&*]{8,}$')
    PT3 = re.compile('^(?=.*[A-Z])(?=.*[!@#$%^&*])[A-Za-z\d!@#$%^&*]{8,}$')
    PT4 = re.compile('^(?=.*[a-z])(?=.*\d)[A-Za-z\d!@#$%^&*]{8,}$')
    PT5 = re.compile('^(?=.*[a-z])(?=.*[!@#$%^&*])[A-Za-z\d!@#$%^&*]{8,}$')
    PT6 = re.compile('^(?=.*\d)(?=.*[!@#$%^&*])[A-Za-z\d!@#$%^&*]{8,}$')

    # 문자 구성 상관없이 10자리 이상 비밀번호 검사 정규식
    PT7 = re.compile('^[A-Za-z\d!@#$%^&*]{10,}$')

    for pattern in [PT1, PT2, PT3, PT4, PT5, PT6, PT7]:
        if pattern.match(password):
            return True
    return False
```

참고로, 위 코드의 특수문자('!@#$%^&*')는 기업 내부 정책에 따라 변경하여 사용하면 되며, 패스워드를 숫자로만 10자리 구성할 경우 취약할 수 있으니 사용자에게 주의할 것을 안내하는 것이 필요하다.

• Django 프레임워크의 VALIDATORS 사용

Django에서는 미들웨어의 AUTH_PASSWORD_VALIDATORS 설정에서 비밀번호에 대한 검증을 하고 있다. 기본적으로 아래와 같은 검증을 해준다.

- UserAttributeSimilarityValidator : user의 attributes(username, firstname, lastname, email)등과 유사한지 체크
- MinimumLengthValidator : 비밀번호 길이의 최소 값 (default 8)
- CommonPasswordValidator : 사람들이 가장 많이 사용하는 패스워드 20,000개에 해당하는지 검증.
- NumericPasswordValidator : 비밀번호가 완전히 숫자인지 검증.

위의 Validator외에 필요한 검증 기준은 사용자 Validator를 생성해서 AUTH_PASSWORD_VALIDATORS에 등록하여 사용가능하다. 아래는 사용자 Validator 예제이다.

(검증 통과 시 None 반환, 실패 시 ValidationError 발생하도록 구현 필요)

안전한 코드의 예

```
import re
from django.core.exceptions import ValidationError
from django.utils.translation import ugettext as _

class CustomValidator(object):
    def validate(self, password, user=None):
        # 2종 이상 문자로 구성된 8자리 이상 비밀번호 검사 정규식
        PT1 = re.compile('^(?=.*[A-Z])(?=.*[a-z])[A-Za-z\d!@#$%^&*]{8,}$')
        PT2 = re.compile('^(?=.*[A-Z])(?=.*\d)[A-Za-z\d$@$!%*?&]{8,}$')
        PT3 = re.compile('^(?=.*[A-Z])(?=.*[!@#$%^&*])[A-Za-z\d!@#$%^&*]{8,}$')
        PT4 = re.compile('^(?=.*[a-z])(?=.*\d)[A-Za-z\d!@#$%^&*]{8,}$')
        PT5 = re.compile('^(?=.*[a-z])(?=.*[!@#$%^&*])[A-Za-z\d!@#$%^&*]{8,}$')
        PT6 = re.compile('^(?=.*\d)(?=.*[!@#$%^&*])[A-Za-z\d!@#$%^&*]{8,}$')

        # 문자 구성 상관없이 10자리 이상 비밀번호 검사 정규식
        PT7 = re.compile('^[A-Za-z\d!@#$%^&*]{10,}$')
        for pattern in [PT1, PT2, PT3, PT4, PT5, PT6, PT7]:
            if pattern.match(password):
                return None
        raise ValidationError(
                _("비밀번호 조합규칙에 적합하지 않습니다.."),
                code='improper_password',
                )

    def get_help_text(self):
        return _(
            "비밀번호는 영문 대문자, 소문자, 숫자, 특수문자 조합 중 2가지 이상 8자리이거나 문자 구성 상
관없이 10자리 이상이어야 합니다."
        )
```

라. 참고자료

① CWE-521: Weak Password Requirements, MITRE,
https://cwe.mitre.org/data/definitions/521.html

② Authentication Cheat Sheet, OWASP,
https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html

③ Regular Expression HOWTO, Python Software Foundation,
https://docs.python.org/3/howto/regex.html

④ Password management in Django, Django Software Foundation,
https://docs.djangoproject.com/en/4.0/topics/auth/passwords/

10. 사용자 하드디스크에 저장되는 쿠키를 통한 정보 노출

가. 개요

대부분의 웹 응용프로그램에서 쿠키는 메모리에 상주하며, 브라우저의 실행이 종료되면 사라진다. 프로그래머가 원하는 경우, 브라우저 세션에 관계없이 지속적으로 저장되도록 설정할 수 있으며, 이것은 디스크에 기록되고, 다음 브라우저 세션이 시작되었을 때 메모리에 로드 된다. 개인정보, 인증 정보 등이 이와 같은 영속적인 쿠키(Persistent Cookie)에 저장된다면, 공격자는 쿠키에 접근할 수 있는 보다 많은 기회를 가지게 되며, 이는 시스템을 취약하게 만든다.

나. 안전한 코딩기법

쿠키의 만료시간은 세션이 지속되는 시간을 고려하여 최소한으로 설정하고 영속적인 쿠키에는 사용자 권한 등급, 세션ID 등 중요정보가 포함되지 않도록 한다.

다. 코드예제

쿠키의 만료시간을 과도하게 길게 설정하면 사용자 하드디스크에 저장된 쿠키는 쉽게 도용될 수 있으므로 취약하다.

안전하지 않은 코드의 예

```
1:  from django.http import HttpResponse
2:
3:  def remind_user_state(request):
4:      res = HttpResponse()
5:      # 쿠키의 만료시간을 1년으로 과도하게 길게 설정하고 있어 안전하지 않다.
6:      res.set_cookie('rememberme', 1, max_age=60*60*24*365)
7:      return res
```

만료시간은 해당 기능에 맞춰 최소로 설정하고 영속적인 쿠키에는 중요 정보가 포함되지 않도록 한다. 쿠키를 HTTPS를 통해서만 전송하도록 secure 속성값을 True(기본값은 False)를 사용할 수 있다. 클라이언트 측에서 JavaScript를 통해 쿠키를 접근하지 못하도록 제한 하고자 할 경우엔 httponly 속성을 True(기본값은 False)로 설정한다.

다음은 쿠키 만료 시간을 1시간으로 설정한 예시이다.

안전한 코드의 예

```
1:  from django.http import HttpResponse
2:
3:  def remind_user_state(request):
4:      res = HttpResponse()
5:      #쿠키의 만료시간을 적절하게 부여하고 secure 옵션을 활성화 한다.
6:      res.set_cookie('rememberme', 1, max_age=60*60, secure=True, httponly=True)
7:      return res
```

Django에서는 settings.py에 아래와 같이 추가하여 전역으로 설정할 수 있다.

안전한 코드의 예

```
1:  # settings.py
2:
3:  SESSION_COOKIE_AGE = 60 * 60        # default 2weeks
4:
5:  SESSION_COOKIE_HTTPONLY = True      # default True
6:
7:  SEESION_COOKIE_SECURE = True        # default False
```

라. 참고자료

① CWE-539: Use of Persistent Cookies Containing Sensitive Information, MITRE, https://cwe.mitre.org/data/definitions/539.html

② Expire and Max-Age Attributes, OWASP, https://cheatsheetseries.owasp.org/cheatsheets/Session_Management_Cheat_Sheet.html#expire-and-max-age-attributes

③ HTTP state management, Python Software Foundation, https://docs.python.org/ko/3/library/http.cookies.html

④ Django set_cookie, Django Software Foundation, https://docs.djangoproject.com/en/dev/ref/request-response/#django.http.HttpResponse.set_cookie

⑤ Django Settings, Django Software Foundation, https://docs.djangoproject.com/en/4.0/ref/settings/#sessions

11. 주석문 안에 포함된 시스템 주요정보

가. 개요

패스워드를 주석문에 넣어두면 시스템 보안이 훼손될 수 있다. 소프트웨어 개발자가 편의를 위해서 주석문에 패스워드를 적어둔 경우, 소프트웨어가 완성된 후에는 그것을 제거하는 것이 매우 어렵게 된다. 또한, 공격자가 소스코드에 접근할 수 있다면, 아주 쉽게 시스템에 침입할 수 있다.

나. 안전한 코딩기법

주석에는 ID, 패스워드 등 보안과 관련된 내용을 기입하지 않는다.

다. 코드예제

편리성을 위해 아이디, 패스워드 등 중요정보를 주석문 안에 작성 후 지우지 않는 경우 정보노출 보안약점이 발생한다.

안전하지 않은 코드의 예

```
1:  def user_login(id, passwd):
2:      # 주석문에 포함된 중요 시스템의 인증 정보
3:      # id = admin
4:      # passwd = passw0rd
5:      result = login(id, passwd)
6:
7:      return result
```

프로그램 개발 시에 주석문 등에 남겨놓은 사용자 계정이나 패스워드 등의 정보는 개발 완료 시에 확실하게 삭제하여야 한다.

안전한 코드의 예

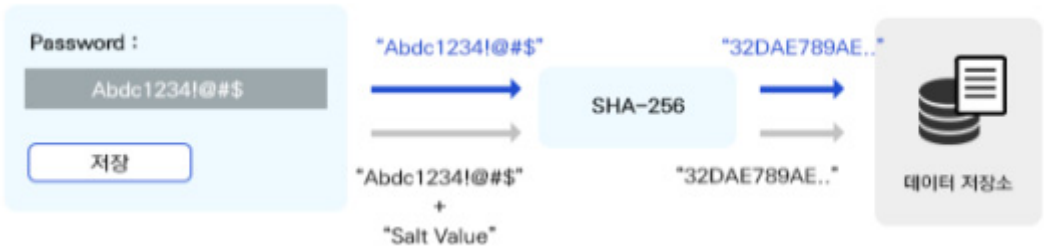
```
1:  def user_login(id, passwd):
2:      # 주석문에 포함된 민감한 정보는 삭제
3:      result = login(id, passwd)
4:      return result
```

라. 참고자료

① CWE-615: Inclusion of Sensitive Information in Source Code Comments, MITRE, https://cwe.mitre.org/data/definitions/615.html

12. 솔트 없이 일방향 해쉬 함수 사용

가. 개요



패스워드와 같은 중요정보를 저장할 경우, 임의의 길이인 데이터를 고정된 크기의 해쉬값으로 변환해주는 일방향 해쉬함수를 이용하여 저장한다. 만약 중요정보를 솔트(Salt)없이 일방향 해쉬함수를 사용하여 저장한다면, 공격자는 미리 계산된 레인보우 테이블을 이용하여 해쉬값을 알아낼 수 있다.

나. 안전한 코딩기법

패스워드와 같은 중요정보를 저장할 경우, 임의의 길이인 데이터를 고정된 크기의 해쉬값으로 변환해주는 일방향 해쉬함수를 이용하여 저장한다.

Python에서 hashlib라이브러리를 사용해서 Hash값을 생성 할 수 있고, salt 값은 os.urandom()등 안전한 난수 생성 라이브러리를 사용하여 생성해야 한다.

다. 코드예제

아래 예시와 같은 코드는 비밀번호와 같은 값을 해쉬하고 있는데 원문의 길이가 짧기 때문에 공격자에 의해 쉽게 유추될 수 있다.

안전하지 않은 코드의 예

```
1:  import hashlib
2:
3:  def get_hash_from_pwd(pw):
4:      #  salt가 없이 생성된 해시값은 강도가 약하기 때문에 반드시 salt와 함께
5:      # 생성해야 한다.
6:      h = hashlib.sha256(pw.encode())
7:
8:      return h.digest()
```

짧은 길이의 비밀번호를 이용하여 강도 높은 해쉬값을 생성하기 위해서는 반드시 seed 값과 함께 해쉬를 해야 한다.

안전한 코드의 예

```
1:  import hashlib
2:  import secrets
3:
4:  def get_hash_from_pwd(pw):
5:      # 비밀번호와 같이 길이가 짧은 값을 이용하여
6:      # 고강도의 해시를 생성하기 위해서는 salt값을
7:      # 사용하면 강도높은 해시를 생성할 수 있다
8:      salt = secrets.token_hex(32)
9:      h = hashlib.sha256(salt.encode() + pw.encode())
10:
11:     return h.digest(), salt
```

라. 참고자료

① CWE-759: Use of a One-Way Hash without a Salt, MITRE,
https://cwe.mitre.org/data/definitions/759.html

② Password Storage Cheat Sheet - Salting, OWASP,
https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#salting

③ hashlib - Secure hashes and message digests, Python Software Foundation,
https://docs.python.org/3/library/hashlib.html

④ secrets, Python Software Foundation,
https://docs.python.org/ko/3/library/secrets.html#module-secrets

13. 무결성 검사없는 코드 다운로드

가. 개요

원격으로부터 소스코드 또는 실행파일을 무결성 검사 없이 다운로드 받고, 이를 실행하는 제품들이 종종 존재한다. 이는 호스트 서버의 변조, DNS 스푸핑(Spoofing) 또는 전송 시의 코드 변조 등의 방법을 이용하여 공격자가 악의적인 코드를 실행할 수 있도록 한다.

파일(및 해당 소프트웨어) 무결성을 확인하는 두 가지 주요 방법으로는 암호화 해시 및 디지털 서명이 있다. 무결성을 보장하기 위해 해시를 사용하고 가능하면 적절한 코드 서명 인증서를 사용하고 확인하는 것이 더 안전하다.

나. 안전한 코딩기법

DNS 스푸핑(Spoofing)을 방어할 수 있는 DNS lookup을 수행하고 코드 전송 시 신뢰할 수 있는 암호 기법을 이용하여 코드를 암호화한다. 또한 다운로드한 코드는 작업 수행을 위해 필요한 최소한의 권한으로 실행하도록 한다.

소스는 신뢰할 수 있는 사이트에서 다운로드하여야 하고 파일의 인증서 또는 해쉬값을 검사하여 변조되지 않은 파일인지 확인하여야 한다.

다. 코드예제

이 예제는 requests.get을 통해 원격에서 파일을 다운로드한 뒤 파일에 대한 무결성 검사를 수행하지 않아 파일변조 등으로 인한 피해가 발생할 수 있는 경우이다. 이러한 경우 공격자는 악의적인 코드를 실행할 수 있다.

안전하지 않은 코드의 예

```
1:  import os
2:  import requests
3:  from urllib.parse import urlparse
4:
5:  def execute_remote_code():
6:      #신뢰되지 않는 사이트에서 코드를 다운로드
7:      url = "https://www.somewhere.com/storage/code.py"
8:
9:      # 원격 코드 다운로드
10:     file = requests.get(url)
11:     remote_code = file.content
12:
13:     file_name = 'save.py'
14:1    with open(file_name, 'wb') as f
5:          f.write(file.content)
16:     ......
```

안전한 코드 실행을 위하여 다운로드한 파일과 해당 파일의 해시값 비교 등을 통해 무결성 검사를 거치고 코드를 실행하여야 한다.

안전한 코드의 예

```
1:  import requests
2:  import hashlib
3:
4:  def execute_remote_code():
5:      url = "https://www.somewhere.com/storage/code.py"
6:      remote_code_hash = config.get('HASH', 'file_hash')
7:
8:      #원격 코드 다운로드
9:      file = requests.get(url)
10:     remote_code = file.content
11:
12:     sha = hashlib.sha256()
13:     sha.update(remote_code)
14:
15:     #다운로드 받은 파일의 해시값 검증
16:     if sha.hexdigest() != remote_code_hash:
17:         raise Exception('파일이 손상되었습니다.')
18:
19:     file_name = 'save.py'
20:     with open(file_name, 'wb') as f
21:         f.write(file.content)
22:     ......
```

라. 참고자료

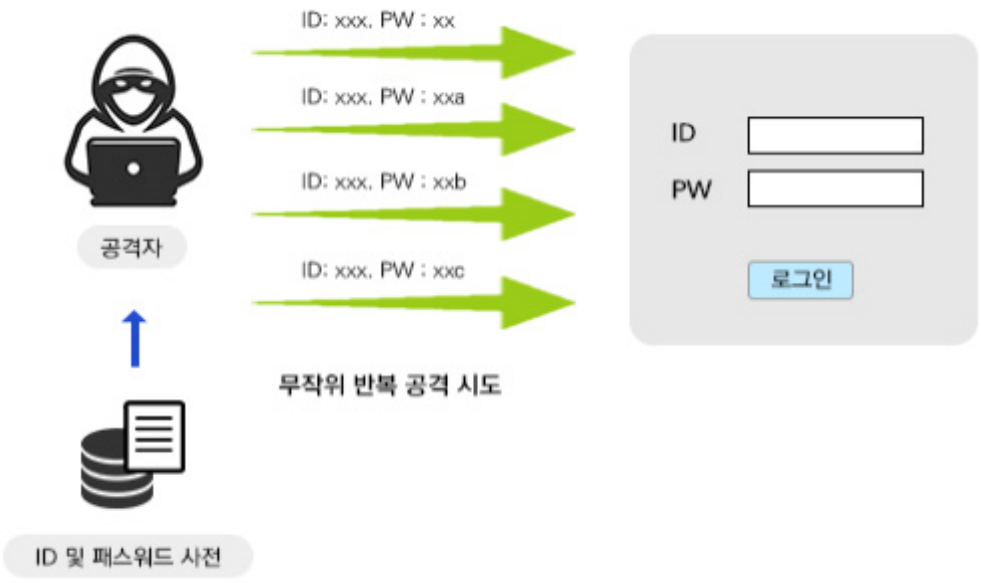
① CWE-494: Download of Code Without Integrity Check, MITRE,
https://cwe.mitre.org/data/definitions/494.html

② Secure hashes and message digests, Python Software Foundation,
https://docs.python.org/3/library/hashlib.html

③ Top 25 Series - Download of Code Without Integrity Check, SANS,
https://www.sans.org/blog/top-25-series-rank-20-download-of-code-without-integrity-check/

14. 반복된 인증시도 제한 기능 부재

가. 개요



일정 시간 내에 여러 번의 인증을 시도하여도 계정 잠금 또는 추가 인증 방법 등의 충분한 조치가 수행되지 않는 경우, 공격자는 성공할법한 ID와 비밀번호들을 사전(Dictionary)으로 만들고 무차별 대입 (brute-force)하여 로그인 성공 및 권한 획득이 가능하다.

Django는 사용자 인증 요청 횟수에 대해 제어하지 않는다. 인증 시스템에 대한 무차별 대입 공격으로부터 보호하기 위해 Django 플러그인(django-defender) 또는 웹 서버 모듈을 사용하여 요청을 제한할 수도 있다.

나. 안전한 코딩기법

인증시도 횟수를 적절한 횟수로 제한하고 설정된 인증실패 횟수를 초과했을 경우 계정을 잠금 하거나 추가적인 인증과정을 거쳐서 시스템에 접근이 가능하도록 한다. 코드 상에서 인증시도 횟수를 제한하는 방법 외에 CAPTCHA나 Two-Factor 인증 방법도 설계 시부터 고려해야 한다.

다. 코드예제

다음 예제는 사용자 로그인 시도에 대한 횟수를 제한하지 않는 코드이다.

안전하지 않은 코드의 예

```
1:  import hashlib
2:  from django.shortcuts import render
3:
4:  def login(request):
5:      user_id = request.POST.get('user_id', '')
6:      user_pw = request.POST.get('user_pw', '')
7:
8:      sha = hashlib.sha256()
9:      sha.update(user_pw)
10:
11:     hashed_passwd = get_user_pw(user_id)
12:
13:     # 인증시도에 따른 제한이 없어 반복적인 인증 시도가 가능
14:     if sha.hexdigest() == hashed_passwd:
15:         return render(request, '/index.html', {'state':'login_success'})
16:     else:
17:         return render(request, '/login.html', {'state':'login_failed'})
```

다음은 사용자 로그인 시도에 대한 횟수를 제한하여 무차별 공격에 대응하는 코드이다.

안전한 코드의 예

```
1:  import hashlib
2:  from django.shortcuts import render
3:
4:  LOGIN_TRY_LIMIT = 5
5:
6:  def login(request):
7:      user_id = request.POST.get('user_id', '')
8:      user_pw = request.POST.get('user_pw', '')
9:
10:     # 로그인 실패기록 가져오기
11:     login_fail = models.LoginFail.objects.filter(user_id)
12:     # 로그인 실패횟수 초과로 인해 잠금된 계정에 대한 인증 시도 제한
13:     if login_fail.count() >= LOGIN_TRY_LIMIT:
14:         return render(request, '/account_lock.html', {'state':'account_lock'})
15:     else:
16:         sha = hashlib.sha256()
17:         sha.update(user_pw)
```

안전한 코드의 예

```
18:
19:             hashed_passwd = get_user_pw(user_id)
20:
21:             if sha.hexdigest() == hashed_passwd:
22:                 models.LoginFail.objects.filter(user_id).delete()
23:                 return render(request, '/index.html', {'state':'login_success'})
24:             else:
25:                 # 로그인 실패횟수 DB 기록
26:                 models.LoginFail.objects.create(user_id)
27:                 return render(request, '/login.html', {'state':'login_failed'})
```

라. 참고자료

① CWE-307: Improper Restriction of Excessive Authentication Attempts, MITRE, https://cwe.mitre.org/data/definitions/307.html

② Blocking Brute Force Attacks, OWASP, https://owasp.org/www-community/controls/Blocking_Brute_Force_Attacks

③ additional security topics, Django Software Foundation, https://docs.djangoproject.com/en/3.2/topics/security/#additional-security-topics

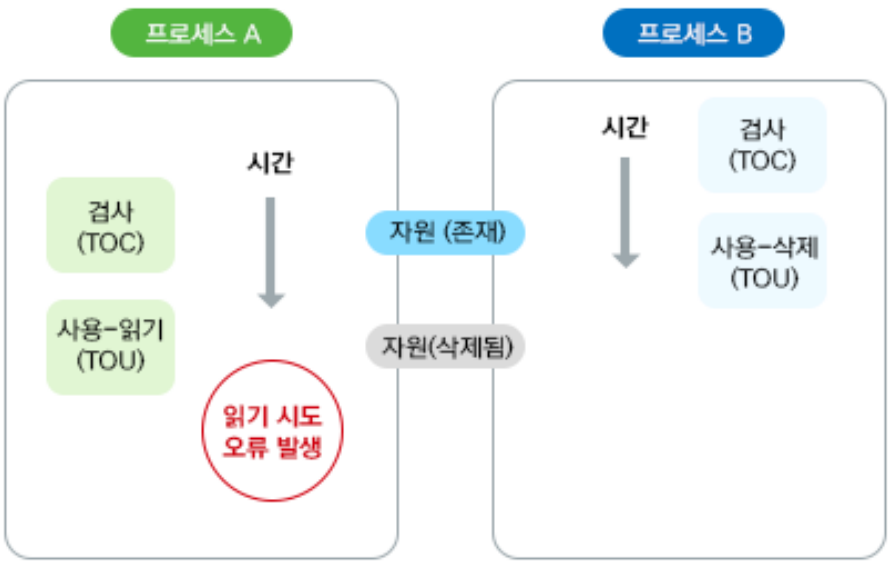
④ Django-defender, https://github.com/jazzband/django-defender

제3절 시간 및 상태

동시 또는 거의 동시 수행을 지원하는 병렬 시스템이나 하나 이상의 프로세스가 동작되는 환경에서 시간 및 상태를 부적절하게 관리하여 발생할 수 있는 보안약점이다.

1. 경쟁조건: 검사시점과 사용시점(TOCTOU)

가. 개요



병렬시스템(멀티프로세스로 구현한 응용프로그램)에서는 자원(파일, 소켓 등)을 사용하기에 앞서 자원의 상태를 검사한다. 하지만, 자원을 사용하는 시점과 검사하는 시점이 다르기 때문에, 검사하는 시점(Time Of Check)에 존재하던 자원이 사용하던 시점(Time Of Use)에 사라지는 등 자원의 상태가 변하는 경우가 발생한다.

예를 들어, 프로세스 A와 B가 존재하는 병렬시스템 환경에서 프로세스 A는 자원사용(파일 읽기)에 앞서 해당 자원(파일)의 존재 여부를 검사(TOC) 한다. 이 때는 프로세스 B가 해당 자원(파일)을 아직 사용(삭제)하지 않았기 때문에, 프로세스 A는 해당 자원(파일)이 존재한다고 판단한다. 그러나 프로세스 A가 자원 사용(파일읽기)을 시도하는 시점(TOU)에 해당 자원(파일)은 사용불가능 상태이기 때문에 오류 등이 발생할 수 있다.

이와 같이 하나의 자원에 대하여 동시에 검사시점과 사용시점이 달라 생기는 보안약점으로 인해 동기화 오류뿐만 아니라 교착상태 등과 같은 문제점이 발생할 수 있다.

Python에서 멀티스레드 환경에서 공유 자원에 여러 쓰레드가 접근하는 것을 막기 위해 Lock 객체를 제공한다. 두 가지 기본 메서드가 제공된다. 자원의 상태를 잠금으로 변경하는 acquire() 메서드와 사용 중인 자원을 해제하는 release()메서드를 제공한다.

나. 안전한 코딩기법

공유자원(예: 파일)을 여러 프로세스가 접근하여 사용할 경우, 동기화 구문을 사용하여 한 번에 하나의 프로세스만 접근 가능하도록 하는 한편, 성능에 미치는 영향을 최소화하기 위해 임계코드 주변만 동기화 구문을 사용한다.

Python의 Lock 객체를 사용 시 lock.acquire()로 자원을 잠그고 lock.release()로 자원을 해제해야 하지만 with문을 사용하여 간단하게 표현도 가능하다.

다. 코드예제

다음 예제는 파일의 검사 후 사용하는 코드가 호출되는 사이의 짧은 시간에도 파일이 사라져서 해당 동작을 할 수 없는 경우가 발생할 수 있다

안전하지 않은 코드의 예

```
1:  import io
2:  import datetime
3:  import threading
4:
5:  def write_shared_file(filename, content):
6:      # 멀티스레드 환경에서는 다른 사용자들의 작업에 따라 파일이 사라질
7:      # 수 있기 때문에 공유 자원에 대해서는 검사와 사용을 동시에 해야 한다.
8:      if os.path.isfile(filename) is True:
9:          f = open(filename, 'w')
10:         f.seek(0, io.SEEK_END)
11:         f.write(content)
12:         f.close()
13:
14: def start():
15:     filename = './temp.txt'
16:     content = 'start time is ' + datetime.datetime.now()
17:     my_thread = threading.Thread(target=write_shared_file, args=(filename, content))
18:     my_thread.start()
```

검사 후 파일이 삭제되거나 변동되는 것을 예방하기 위해 lock을 사용하여 각 쓰레드에서 공유자원에 접근하는 것을 통제하는 예제이다. lock을 acquire하면 해당 쓰레드만 공유 데이터에 접근 할 수 있고, lock을 release 해야만 다른 쓰레드에서 공유 데이터에 접근 할 수 있다.

안전한 코드의 예

```
1:  import io
2:  import datetime
3:  import threading
4:
5:  lock = threading.Lock()
6:  def write_shared_file(filename, content):
7:      # lock을 이용하여 여러 사용자가 동시에 파일에 접근하지 못하도록 제한
8:      with lock:
9:          if os.path.isfile(filename) is True:
10:             f = open(filename, 'w')
11:             f.seek(0, io.SEEK_END)
12:             f.write(content)
13:             f.close()
14:
15: def start():
16:     filename = './temp.txt'
17:     content = 'start time is ' + datetime.datetime.now()
18:     my_thread = threading.Thread(target=write_shared_file, args=(filename, content))
19:     my_thread.start()
```

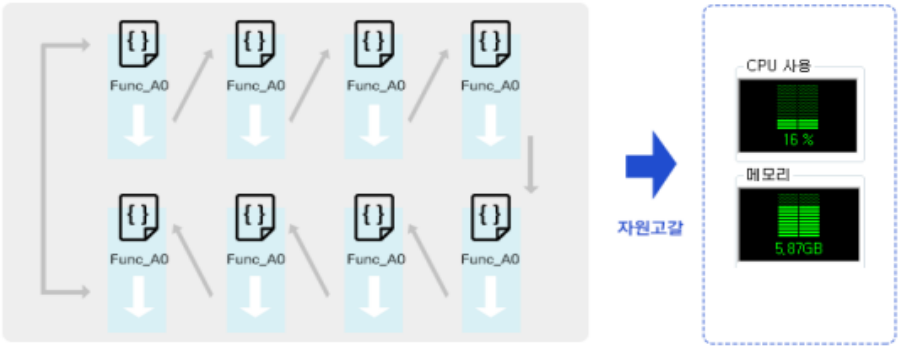
라. 참고자료

① CWE-367: Time-of-check Time-of-use (TOCTOU) Race Condition, MITRE, https://cwe.mitre.org/data/definitions/367.html

② Thread-based parallelism, Python Software Foundation, https://docs.python.org/3/library/threading.html

2. 종료되지 않는 반복문 또는 재귀 함수

가. 개요



재귀의 순환 횟수를 제어하지 못하여 할당된 메모리나 프로그램 스택 등의 자원을 과다하게 사용하면 위험하다. 대부분의 경우, 귀납 조건(Base Case)이 없는 재귀 함수는 무한 루프에 빠져들게 되고 자원고갈을 유발함으로써 시스템의 정상적인 서비스를 제공할 수 없게 한다.

Python에서는 최대 재귀 깊이가 적용되어 있어 무한루프가 발생하지 않으나, setrecursionlimit()를 사용하여 임의로 최대 재귀 깊이를 변경하여 사용하는 경우 과도하게 많이 재귀 함수가 수행되지 않도록 주의하여야 한다.

나. 안전한 코딩기법

모든 재귀 호출 시, 재귀 호출 횟수를 제한하거나, 초기 값을 설정(상수)하여 재귀 호출을 제한해야 한다. Python의 recursionlimit 제한은 stack overflow를 막기 위한 방법이다. recursionlimit 값을 과도하게 큰 값으로 설정하지 않아야 한다.

다. 코드예제

factorial 함수는 함수 내부에서 자신을 호출하는 함수로, 재귀문을 빠져 나오는 조건을 정의하고 있지 않아 시스템 장애를 유발할 수 있다.

안전하지 않은 코드의 예

```
    def factorial(num):
1:      # 재귀함수 탈출조건을 설정하지 않아 동작 중 에러 발생
2:      return num * factorial(num - 1)
3:
4:  if __name__ == '__main__':
5:      itr = 5
6:      result = factorial(itr)
7:      print(str(itr) + ' 팩토리얼 값은 : ' + str(result))
```

무한루프를 사용하지 않고 특정 조건 또는 횟수에 따라 동작한다.

안전한 코드의 예

```
1:  def factorial(num):
2:      # 재귀함수 사용 시에는 탈출 조건을 명시해야 한다.
3:      if (num == 0):
4:          return 1
5:      else:
6:          return num * factorial(num - 1)
7:
8:  if __name__ == '__main__':
9:      itr = 5
10:     result = factorial(itr)
11:     print(str(itr) + ' 팩토리얼 값은 : ' + str(result))
```

Python의 재귀 반복 제한(Recursion Depth Limit)은 기본이 1000으로 설정되어 있다.

Anaconda의 경우는 기본 값이 2000이다. 이 값을 과도하게 변경하지 않아야 한다.

안전한 코드의 예

```
1:  import sys
2:
3:  sys.setrecursionlimit(1000)
```

라. 참고자료

① CWE-674: Uncontrolled Recursion, MITRE,
https://cwe.mitre.org/data/definitions/674.html

② CWE-835: Loop with Unreachable Exit Condition ('Infinite Loop'), MITRE,
https://cwe.mitre.org/data/definitions/835.html

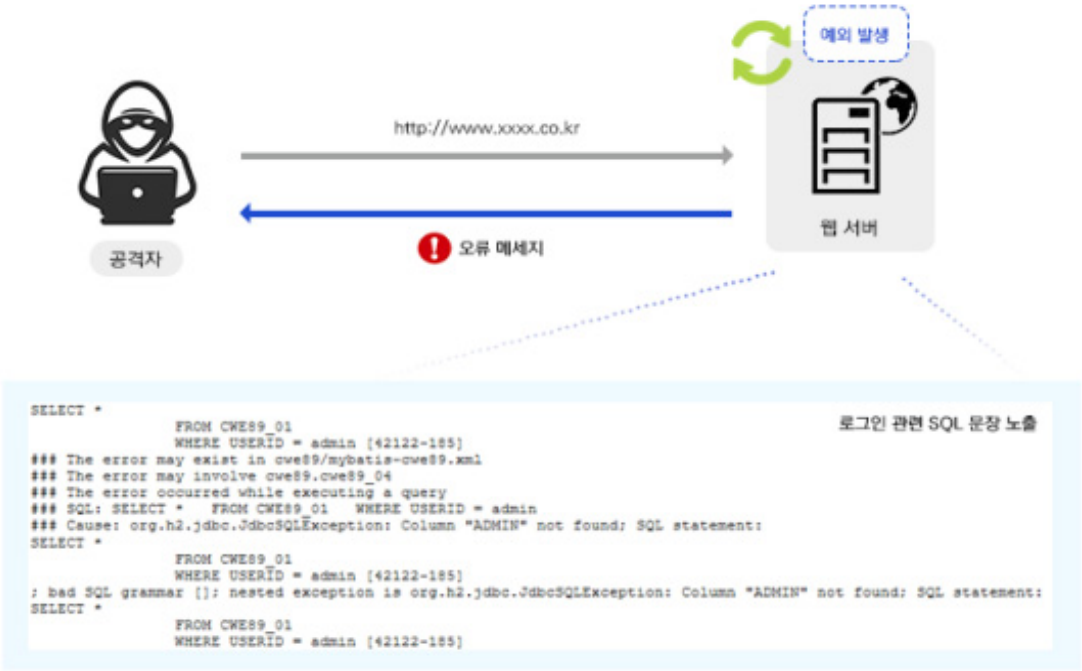
③ sys.setrecursionlimit, Python Software Foundation,
https://docs.python.org/3/library/sys.html#sys.setrecursionlimit

제4절 에러처리

에러를 처리하지 않거나, 불충분하게 처리하여 에러 정보에 중요정보(시스템 내부정보 등)가 포함될 때, 발생할 수 있는 취약점으로 에러를 부적절하게 처리하여 발생하는 보안약점이다.

1. 오류 메시지 정보노출

가. 개요



응용프로그램이 실행환경, 사용자 등 관련 데이터에 대한 민감한 정보를 포함하는 오류 메시지를 생성하여 외부에 제공하는 경우, 공격자의 악성 행위를 도울 수 있다. 예외발생 시 예외 이름이나 트레이스백(traceback)를 출력하는 경우, 프로그램 내부 구조를 쉽게 파악할 수 있기 때문이다.

Django 프레임워크와 Flask 프레임워크는 HTTP 오류 코드가 있는 요청을 처리하기 위한 사용자 에러 페이지 핸들러를 제공한다.

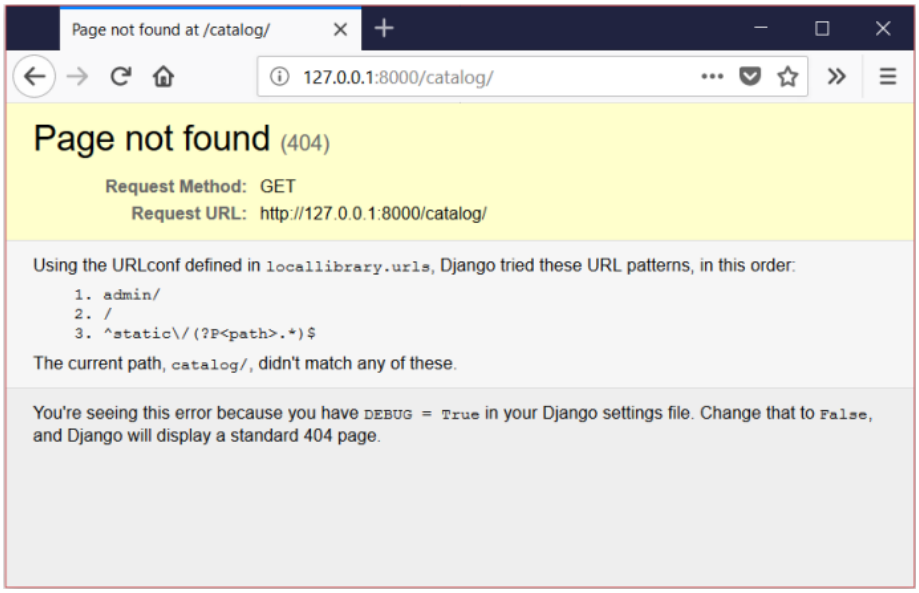
나. 안전한 코딩기법

오류 메시지는 정해진 사용자에게 유용한 최소한의 정보만 포함하도록 한다. 소스코드에서 예외 상황은 내부적으로 처리하고 사용자에게 시스템 내부 정보 등 민감한 정보를 포함하는 오류를 출력하지 않도록 미리 정의된 메시지를 제공하도록 설정한다.

Django 프레임워크에서는 urls.py에 사용자 정의 에러 페이지를 핸들러를 정의 할 수 있다.

다. 코드예제

사용자 요청을 처리할 수 없는 경우 에러 페이지에 디버그 정보 또는 서버의 정보가 노출될 수 있다. 만일 어플리케이션을 배포 할 때 DEBUG 모드를 True로 설정 하고 배포할 경우에 아래와 같이 시스템의 주요 정보가 노출 될 수도 있다.



Django는 DEBUG 모드를 False로 배포했을 경우 아래와 같이 사용자 에러페이지를 설정하지 않을 경우에는 Django 기본 에러페이지가 보이게 된다.

안전하지 않은 코드의 예

```
1:  # config/urls.py
2:  # 별도 에러페이지를 선언하지 않아 django의 기본 에러페이지를 출력한다.
```

제공되는 에러 페이지 핸들러를 이용해 별도의 에러 페이지를 생성하여 사용자에게 표현하고 서버의 정보노출을 최소화 한다.

안전한 코드의 예

```
1:  # config/urls.py
2:  from django.conf.urls import handler400, handler403, handler404, handler500
3:
4:  # 사용자 정의 에러페이지를 지정하고
5:  # views.py에 사용자 정의 에러페이지에 대한 코드를 구현하여 사용한다.
6:  handler400 = "blog.views.error400"
7:  handler403 = "blog.views.error403"
8:  handler404 = "blog.views.error404"
9:  handler500 = "blog.views.error500"
```

아래는 traceback을 사용하여 에러 스택을 표준출력으로 표시하는 예제이다.

안전하지 않은 코드의 예

```
1:  import traceback
2:
3:  def fetch_url(url, useragent, referer=None, retries=1, dimension=False):
4:      ......
5:      try:
6:          response = requests.get(url, stream=True, timeout=5, headers={
7:              'User-Agent': useragent,
8:              'Referer': referer,
9:          })
10:         ......
11:     except IOError:
12:         # 에러메시지를 통해 스택정보가 노출됨.
13:         traceback.print_exc()
14:         break
```

오류 처리 시 아래와 같이 에러 이름이나 에러 추적 정보가 노출 되지 않도록 한다.

안전한 코드의 예

```
1:  import logging
2:
3:  def fetch_url(url, useragent, referer=None, retries=1, dimension=False):
4:      ......
5:      try:
6:          response = requests.get(url, stream=True, timeout=5, headers={
7:              'User-Agent': useragent,
8:              'Referer': referer,
9:          })
10:         ......
11:     except IOError:
12:         # 에러 코드와 정보를 별도로 정의하고 최소 정보만 로깅
13:         logger.info('ERROR-01:통신에러')
14:         break
```

라. 참고자료

① CWE-209: Generation of Error Message Containing Sensitive Information, MITRE
https://cwe.mitre.org/data/definitions/209.html

② Improper Error Handling, OWASP,
https://owasp.org/www-community/Improper_Error_Handling

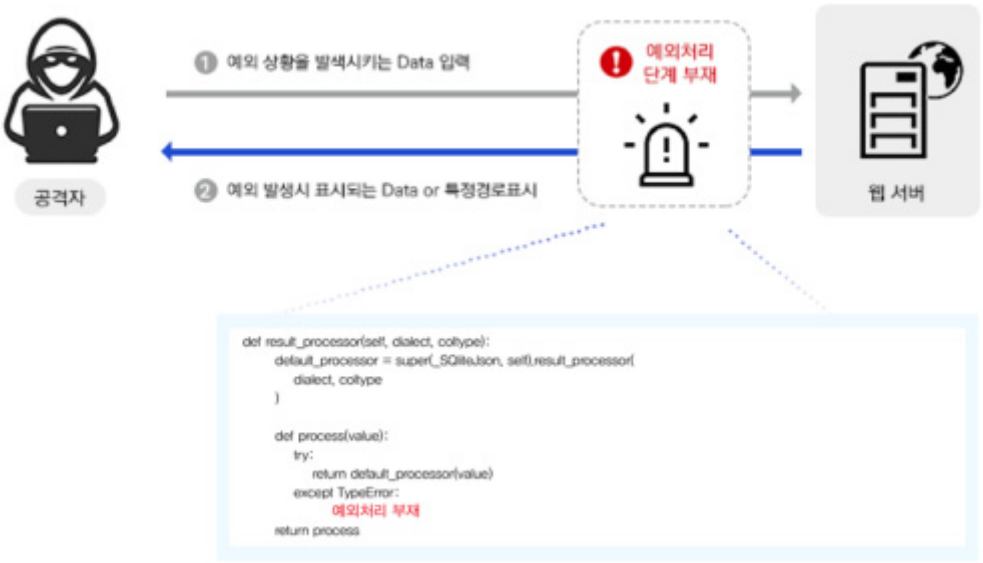
③ Errors and Exceptions, Python Software Foundation,
https://docs.python.org/3/tutorial/errors.html

④ Django Error views, Django Software Foundation,
https://docs.djangoproject.com/en/3.2/ref/views/#error-views

⑤ Flask Error Handlers, Flask
https://flask.palletsprojects.com/en/2.0.x/errorhandling/#error-handlers

2. 오류상황 대응 부재

가. 개요



오류가 발생할 수 있는 부분을 확인하였으나, 이러한 오류에 대하여 예외 처리를 하지 않을 경우, 공격자는 오류 상황을 악용하여 개발자가 의도하지 않은 방향으로 프로그램이 동작하도록 할 수 있다.

예외처리는 코드를 견고하게 만들고 프로그램이 제어되지 않는 방식으로 중지되도록 하는 잠재적인 오류를 방지하는데 도움이 된다.

나. 안전한 코딩기법

오류가 발생할 수 있는 부분에 대하여 제어문(try-except)을 사용하여 적절하게 예외 처리 한다.

다. 코드예제

다음 예제는 try 블록에서 발생하는 오류를 포착(except)하고 있지만, 그 오류에 대해서 아무 조치를 하고 있지 않음을 보여준다. 아무 조치가 없으므로 프로그램이 계속 실행되기 때문에 개발자가 의도하지 않은 방향으로 프로그램이 동작될 수 있다.

안전하지 않은 코드의 예

```
1:  import os
2:  import binascii
3:  import base64
4:  from Crypto.Cipher import AES
5:
6:  static_keys=[
7:              {'key' : b'\xb9J\xfd\xa9\xd2\xefD\x0b\x7f\xb2\xbcy\x9c\xf7\x9c',
8:                'iv'   : b'\xf1BZ\x06\x03TP\xd1\x8a\xad"\xdc\xc3\x08\x88\xda'},
9:              {'key' : b'Z\x01$.:\xd4u3~\xb6TS(\x08\xcc\xfc',
10:               'iv'   : b'\xa1a=:\xba\xfczv]\xca\x83\x9485\x14\x17'},
11: ]
12: def encryption(key_id, plain_text):
13:       static_key = {'key':b'0000000000000000', 'iv':b'0000000000000000'}
14:
15:       try:
16:             static_key = static_keys[key_id]
17:       except IndexError:
18:             # key 선택 중 오류 발생 시 기본으로 설정된 암호화 키인
19:             # '0000000000000000' 으로 암호화가 수행된다.
20:             pass
21:
22:       cipher_aes = AES.new(static_key['key'],AES.MODE_CBC,static_key['iv'])
23:       encrypted_data = base64.b64encode(cipher_aes.encrypt(plain_text))
24:       return encrypted_data.decode('ASCII')
```

예외상황 발생 시에 프로그램이 개발자의 의도와 다르게 동작하지 않도록 반드시 예외를 처리해야 한다.

안전한 코드의 예

```
1:  import os
2:  import binascii
3:  import base64
4:  from Crypto.Cipher import AES
5:
6:  static_keys=[
7:          {'key' : b'\xb9J\xfd\xa9\xd2\xefD\x0b\x7f\xb2\xbcy\x9c\xf7\x9c',
8:            'iv'   : b'\xf1BZ\x06\x03TP\xd1\x8a\xad"\xdc\xc3\x08\x88\xda'},
9:          {'key' : b'Z\x01$.:\xd4u3~\xb6TS(\x08\xcc\xfc',
10:           'iv'   : b'\xa1a=:\xba\xfczv]\xca\x83\x9485\x14\x17'},
11: ]
12: def encryption(key_id, plain_text):
13:     static_key = {'key':b'0000000000000000', 'iv':b'0000000000000000'}
14:
15:     try:
16:         static_key = static_keys[key_id]
17:     except IndexError:
18:         # key 선택 중 오류 발생 시 랜덤으로 암호화 키를 생성하도록 설정
19:         static_key = {'key': secrets.token_bytes(16), 'iv': secrets.token_bytes(16)}
20:         static_keys.append(static_key)
21:
22:     cipher_aes = AES.new(static_key['key'],AES.MODE_CBC,static_key['iv'])
23:     encrypted_data = base64.b64encode(cipher_aes.encrypt(plain_text))
24:     return encrypted_data.decode('ASCII')
```

라. 참고자료

① CWE-390: Detection of Error Condition Without Action, MITRE, https://cwe.mitre.org/data/definitions/390.html

② Errors and Exceptions, Python Software Foundation, https://docs.python.org/3/tutorial/errors.html

③ Built-in Exceptions, Python Software Foundation, https://docs.python.org/3/library/exceptions.html

3. 부적절한 예외 처리

가. 개요

프로그램 수행 중에 함수의 결과 값에 대한 적절한 처리 또는 예외 상황에 대한 조건을 적절하게 검사 하지 않을 경우, 예기치 않은 문제를 야기할 수 있다.

나. 안전한 코딩기법

값을 반환하는 모든 함수의 결과값을 검사하여, 그 값이 의도했던 값인지 검사하고, 예외 처리를 사용 하는 경우에 광범위한 예외 처리 대신 구체적인 예외 처리를 수행한다.

다. 코드예제

다음 예제는 다양한 예외가 발생할 수 있음에도 불구하고 광범위한 예외 처리로 예외상황에 따른 적절한 조치를 처리 할 수 없다.

안전하지 않은 코드의 예

```
1:  import sys
2:
3:  def get_content():
4:      try:
5:          f = open('myfile.txt')
6:          s = f.readline()
7:          i = int(s.strip())
8:      # 예외처리를 세분화 할 수 있음에도 광범위하게 사용하여 예기치 않은
9:      # 문제가 발생할 수 있다.
10:     except:
11:         print("Unexpected error ")
```

발생 가능한 예외를 세분화하여 예외상황에 따라 적합한 예외를 처리한다.

안전한 코드의 예

```
1:  def get_content():
2:      try:
3:          f = open('myfile.txt')
4:          s = f.readline()
5:          i = int(s.strip())
6:      # 발생할 수 있는 오류의 종류와 순서에 맞춰서 예외 처리 한다.
7:      except FileNotFoundError:
8:          print("file is not found")
9:      except OSError:
10:         print("cannot open file")
11:     except ValueError:
12:         print("Could not convert data to an integer.")
```

라. 참고자료

① CWE-754: Improper Check for Unusual or Exceptional Conditions, MITRE, https://cwe.mitre.org/data/definitions/754.html

② Errors and Exceptions, Python Software Foundation, https://docs.python.org/3/tutorial/errors.html

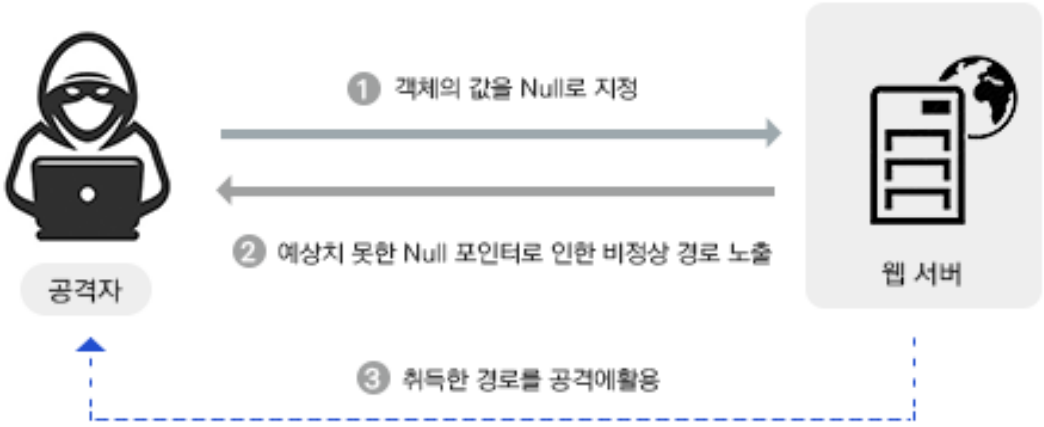
③ Built-in Exceptions, Python Software Foundation, https://docs.python.org/3/library/exceptions.html

제5절 코드오류

타입 변환 오류, 자원(메모리 등)의 부적절한 반환 등과 같이 개발자가 범할 수 있는 코딩 오류로 인해 유발되는 보안 약점이다.

1. Null Pointer 역참조

가. 개요



널 포인터(Null Pointer) 역참조는 '일반적으로 그 객체가 널(Null)이 될 수 없다'라고 하는 가정을 위반했을 때 발생한다. 공격자가 의도적으로 널 포인터 역참조를 발생시키는 경우, 그 결과 발생하는 예외 상황을 이용하여 추후의 공격을 계획하는 데 사용될 수 있다.

Python에서는 Null pointer dereference가 발생하지 않는다. Python에서는 Null 객체가 사용되지 않으며 대신 None 키워드를 사용하여 null 개체와 변수를 정의 한다. None은 다른 언어의 null과 동일한 기능을 수행하지 않으며 None이 0 또는 다른 값을 정의 하진 않는다.

나. 안전한 코딩기법

특별한 의미를 표시하는 None을 반환하는 함수를 사용하면 None과 다른 값(예: 0이나 빈 문자열)이 조건문에서 False로 평가될 수 있기 때문에 실수하기 쉽다.

None이 될 수 있는 개체를 참조하기 전에 None 값인지를 검사하여 시스템 오류를 줄일 수 있다.

다. 코드예제

Python에서 포인터를 사용하지는 않지만 Null pointer와 유사한 None값을 참조하는 오류를 범할 수 있다.

안전하지 않은 코드의 예

```
1:  from django.shortcuts import render
2:  from xml.sax import make_parser, handler
3:  import os
4:
5:  def parse_xml(request):
6:      filename = request.POST.get('filename')
7:      # filename의 None 체크를 하지 않아 에러 발생 가능
8:      if (filename.count('.') > 0):
9:          name, ext = os.path.splitext(filename)
10:     else:
11:         ext = ''
12:
13:     if ext == '.xml':
14:         parser = make_parser()
15:         ......
16:         sax_handler = Handler()
17:         parser.setContentHandler(sax_handler)
18:         parser.parse(filename)
19:
20:         return sax_handler.root
```

참조하고자 하는 자원을 호출할 때에는 개체가 None이 아닌지 검증하여야 한다.

안전한 코드의 예

```
1:  from django.shortcuts import render
2:  from xml.sax import make_parser, handler
3:  import os
4:
5:  def parse_xml(request):
6:      filename = request.POST.get('filename')
7:      # filename이 None 인지 체크
8:      if filename is None or filename.strip() == "":
9:          raise ValueError("파일 이름이 없습니다.")
10:
11:     if (filename.count('.') > 0):
12:         name, ext = os.path.splitext(filename)
13:     else:
14:         ext = ''
15:
16:     if ext == '.xml':
17:         parser = make_parser()
18:         ......
19:         sax_handler = Handler()
20:         parser.setContentHandler(sax_handler)
21:         parser.parse(filename)
22:
23:         return sax_handler.root
```

라. 참고자료

① CWE-476: NULL Pointer Dereference, MITRE,
https://cwe.mitre.org/data/definitions/476.html

② Null Dereference, OWASP,
https://owasp.org/www-community/vulnerabilities/Null_Dereference

③ Built-in Constants, Python Software Foundation,
https://docs.python.org/3/library/constants.html?#None

2. 부적절한 자원 해제

가. 개요

프로그램의 자원, 예를 들면 열린 파일디스크립터(Open File Descriptor), 힙 메모리(Heap Memory), 소켓(Socket) 등은 유한한 자원이다. 이러한 자원을 할당 받아 사용한 후, 더 이상 사용하지 않는 경우 에는 적절히 반환하여야 하는데, 프로그램 오류 또는 에러로 사용이 끝난 자원을 반환하지 못하는 경우이다.

나. 안전한 코딩기법

자원을 획득하여 사용한 다음에는 반드시 자원을 해제하여 반환한다.

다. 코드예제

try구문 내 처리 중 오류가 발생할 경우, close() 메소드가 실행되지 않아 사용한 자원이 반환되지 않을 수 있다.

안전하지 않은 코드의 예

```
1:  import sys
2:  def get_config():
3:      lines = None
4:      try:
5:          f = open('config.cfg')
6:          lines = f.readlines()
7:          ...
8:          # try 절에서 할당한 자원이 반환(close)되기 전에
9:          # 예외가 발생하면 할당된 자원이 시스템에 반환되지 않음
10:         f.close()
11:         return lines
12:     except Exception as e:
13:         ...
14:         return ''
```

예외상황이 발생하여 함수가 종료될 때, 예외의 발생 여부와 상관없이 항상 실행되는 finally 블록에서 할당 받은 모든 자원을 반드시 반환하도록 한다.

안전한 코드의 예

```
1:  import sys
2:
3:  def get_config():
4:      lines = None
5:      try:
6:          f = open('config.cfg')
7:          lines = f.readlines()
8:      except Exception as e:
9:          ...
10:     finally:
11:         # try 절에서 할당한 자원은
12:         # finally절에서 시스템에 반환을 해 주어야 한다
13:         f.close()
14:         return lines
```

다른 방법은 with문을 사용하여 파일을 처리하는 방법으로, with문의 블록이 끝날 때 자동으로 파일을 닫아줍니다. 이 방식은 다른 예외가 발생하더라도 파일 닫기가 보장된다.

안전한 코드의 예

```
1:  # with 절을 빠져나갈 때 f를 시스템에 반환
2:  with open('filename.py') as f:
3:      print(f.read())
```

라. 참고자료

① CWE-404: Improper Resource Shutdown or Release, MITRE,
https://cwe.mitre.org/data/definitions/404.html

② Unreleased Resource, OWASP,
https://owasp.org/www-community/vulnerabilities/Unreleased_Resource

③ The With statement, Python Software Foundation,
https://docs.python.org/3/reference/compound_stmts.html#grammar-token-python-grammar-with_stmt

3. 신뢰할 수 없는 데이터의 역직렬화

가. 개요

직렬화(Serialization)는 프로그램에서 특정 클래스의 현재 인스턴스 상태를 다른 서버로 전달하기 위해 클래스의 인스턴스 정보를 바이트 스트림으로 복사하는 작업으로, 메모리상에서 실행되고 있는 객체의 상태를 그대로 복제하여 파일로 저장하거나 수신 측에 전달하게 된다.

역직렬화(Deserialization)는 반대 연산으로 바이너리 파일(Binary File) 이나 바이트 스트림(Byte Stream)으로부터 객체 구조로 복원하게 된다. 이 때, 송신자가 네트워크를 이용하여 직렬화된 정보를 수신자에게 전달하는 과정에서 공격자가 전송 또는 저장된 스트림을 조작할 수 있는 경우에는 신뢰할 수 없는 역직렬화를 이용하여 무결성 침해, 원격 코드 실행, 서비스 거부 공격 등이 발생 할 수 있는 보안약점이다.

Python에서는 pickle 모듈을 통해 직렬화 및 역직렬화를 지원하고 각각 pickle, unpickle로 명칭한다. pickle 모듈은 데이터 변조에 대한 검증 과정이 없기 때문에 임의의 코드를 실행하는 악의적인 pickle 데이터를 구성할 수 있어 pickle을 사용하여 역직렬화 하는 경우 hmac으로 데이터에 서명하거나, json 모듈을 사용하는 것을 고려해야 한다.

나. 안전한 코딩기법

초기화되지 않은 스택 메모리 영역의 변수는 임의 값이라 생각해서 대수롭지 않게 생각할 수 있으나 사실은 이전 함수에서 사용되었던 내용을 포함하고 있다. 공격자는 이러한 약점을 사용하여 메모리에 저장되어 있는 값을 읽거나 특정 코드를 실행할 수 있다. 모든 변수를 사용 전에 반드시 올바른 초기 값을 할당함으로서 이러한 문제를 예방한다.

신뢰할 수 없는 데이터를 역직렬화 하지 않도록 응용프로그램을 구성한다. 민감정보 또는 중요정보를 전송 시 암호화 통신을 적용하지 못하는 경우, 송신 측에서 서명을 추가하고 수신 측에서 서명을 확인하여 데이터의 무결성을 검증한다. 또는, 신뢰할 수 있는 데이터의 식별을 위해 역직렬화 대상의 데이터가 사전에 검증된 클래스(Class)만을 포함하는지 검증하거나, 제한된 실행 권한을 구성하여 역직렬화 코드를 실행한다.

다. 코드예제

다음 예제는 알 수 없는 사용자로부터 입력 받은 코드를 역직렬화 하고 있는데, 이와 같은 코드는 개발자가 의도하지 않은 임의 코드가 실행될 수 있다.

안전하지 않은 코드의 예

```
1:  import pickle
2:  from django.shortcuts import render
3:
4:  def load_user_object(request):
5:      userinfo = bytes(request.POST.get('userinfo', ''), encoding = "utf-8")
6:      # 사용자로부터 입력받은 알 수 없는 데이터를 직렬화
7:      user_obj = pickle.loads(userinfo)
8:      return render(request, '/load_user_obj.html', {'obj':user_obj})
```

아래 예제는 사용자로부터 전달받은 데이터를 hmac을 이용하여 안전한 사용자로부터 온 것인지 검증한 후 역직렬화 하고 있다.

안전한 코드의 예

```
1:  import hmac
2:  import hashlib
3:  import pickle
4:  from django.shortcuts import render
5:
6:  def load_user_object(request):
7:      hash_pickle = bytes(request.POST.get('hash_pickle', ''), encoding = "utf-8")
8:      userinfo = bytes(request.POST.get('userinfo', ''), encoding = "utf-8")
9:      # HMAC 검증을 위한 비밀키는 안전하게 저장하여 사용
10:     m = hmac.new(key=b'secret_key', digestmod=hashlib.sha512)
11:     m.update(userinfo)
12:     # 전달받은 해시값(hash_pickle)과 직렬화 데이터(userinfo)의
13:     # 해시값을 비교하여 검증
14:     if hmac.compare_digest(m.digest(), hash_pickle):
15:         user_obj = pickle.loads(userinfo_pickle)
16:         return render(request, '/load_user_obj.html', {'obj':user_obj})
17:     else:
18:         return render(request, '/error.html', {'error':'직렬화 오류입니다.'})
```

이 외에도 내부의 데이터만을 필요로 하는 경우에는 JSON과 같은 텍스트 형태의 안전한 직렬화 형식을 사용하는 것이 옳다.

라. 참고자료

① CWE-502: Deserialization of Untrusted Data, MITRE,
https://cwe.mitre.org/data/definitions/502.html

② Deserialization Cheat Sheet, OWASP,
https://cheatsheetseries.owasp.org/cheatsheets/Deserialization_Cheat_Sheet.html

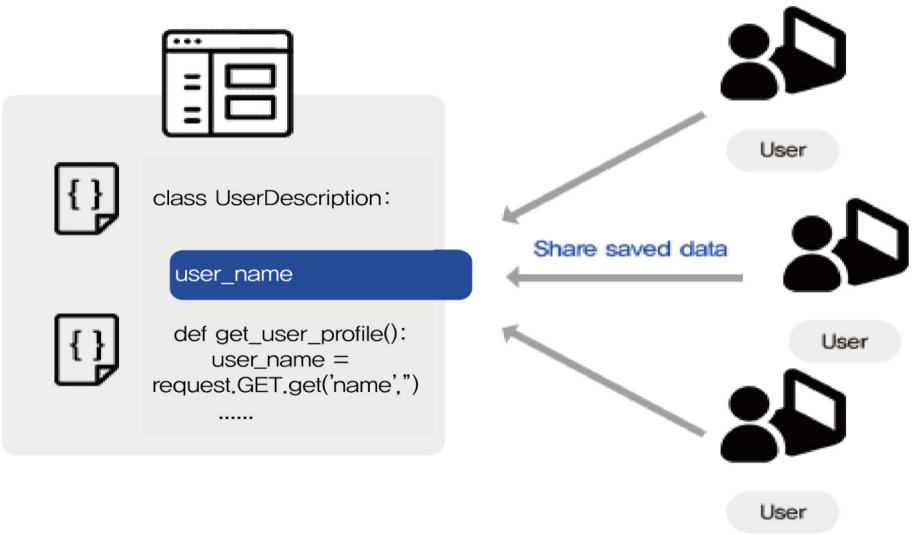
③ Python object serialization, Python Software Foundation,
https://docs.python.org/3/library/pickle.html

제6절 캡슐화

중요한 데이터 또는 기능성을 불충분하게 캡슐화하거나 잘못 사용함으로써 발생하는 보안약점으로 정보노출, 권한 문제 등이 발생할 수 있다.

1. 잘못된 세션에 의한 데이터 정보 노출

가. 개요



다중 스레드 환경에서는 싱글톤(Singleton) 객체 필드에 경쟁조건(Race Condition)이 발생할 수 있다. 따라서, 다중 스레드 환경에서는 정보를 저장하는 전역 변수가 포함되지 않도록 하여, 서로 다른 세션에서 데이터를 공유하지 않도록 해야 한다.

나. 안전한 코딩기법

싱글톤 패턴을 사용하는 경우, 변수 범위(Scope)에 주의를 기울여야 한다. 특히 다중 스레드 환경에서 클래스 변수의 값은 하위 메소드와 공유되므로 필요한 경우 인스턴스 변수로 선언하여 사용한다.

다. 코드예제

다중 스레드 환경에서 파이썬의 클래스 변수는 스레드 간 서로 공유하게 된다.

클래스 변수에 값을 할당할 경우 서로 다른 세션 간에 데이터가 공유되어 의도하지 않은 데이터가 전달될 수 있다.

안전하지 않은 코드의 예

```
1:  from django.shortcuts import render
2:
3:  class UserDescription:
4:      user_name = ''
5:
6:      def get_user_profile(self):
7:          result = self.get_user_discription(UserDescription.user_name)
8:          ......
9:          return result
10:
11:     def show_user_profile(self, name):
12:         # 클래스변수는 다른 세션과 공유되는 값이기 때문에 멀티스레드
13:         # 환경에서 공유되지 말아야할 자원에 대해서 사용하는 경우
14:         # 다른 세션에 의해 데이터가 노출될 수 있다.
15:         UserDescription.user_name = name
16:         self.user_profile = self.get_user_profile()
17:
18:         return render(request, 'profile.html', {'profile':self.user_profile})
```

공유되어서는 안 되는 변수는 인스턴스 변수로 선언하여 세션 간 공유 되지 않도록 한다.

안전한 코드의 예

```
1:  from django.shortcuts import render
2:
3:  class UserDescription:
4:
5:      def get_user_profile(self):
6:          result = self.get_user_discription(self.user_name)
7:          ......
8:          return result
9:
10:     def show_user_profile(self, name):
11:         # 인스턴스 변수로 사용하여 스레드 간 공유되지 못하도록 한다.
12:         self.user_name = name
13:         self.user_profile = self.get_user_profile()
14:
15:         return render(request, 'profile.html', {'profile':self.user_profile})
```

라. 참고자료

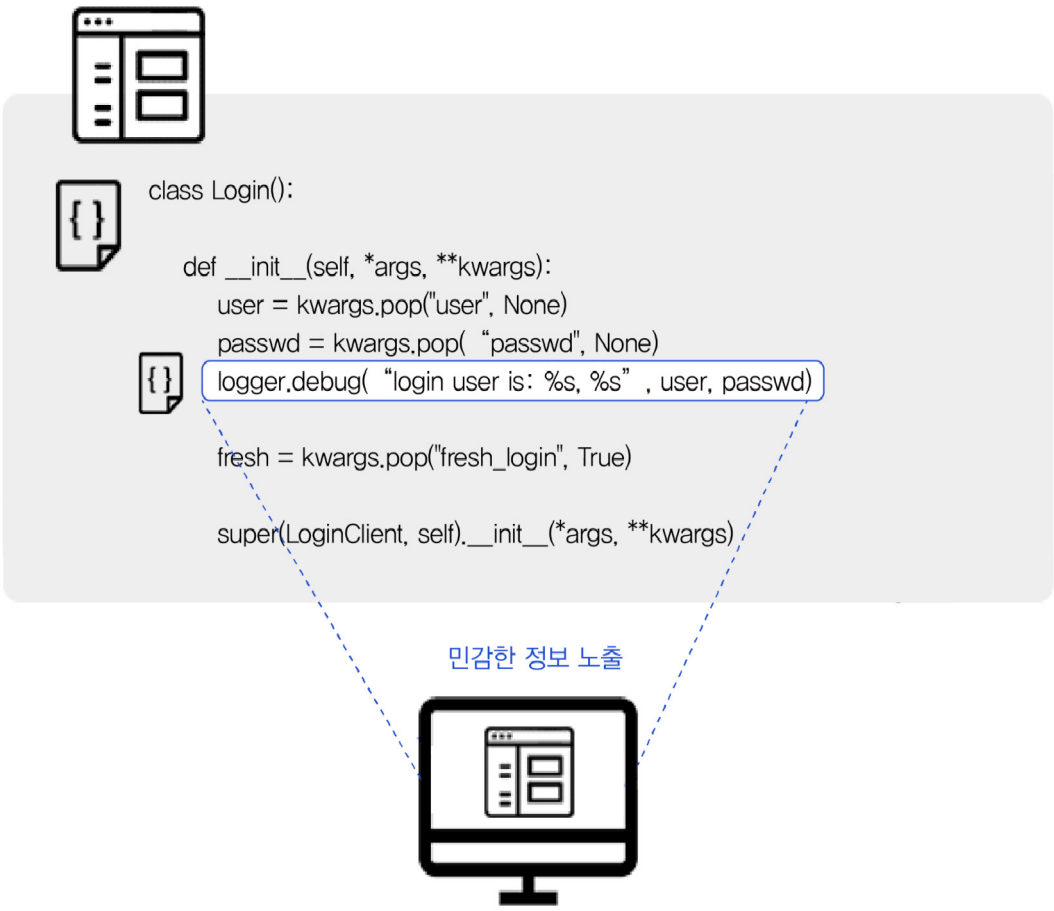
① CWE-488: Exposure of Data Element to Wrong Session, MITRE, https://cwe.mitre.org/data/definitions/488.html

② CWE-543: Use of Singleton Pattern Without Synchronization in a Multithreaded Context, MITRE, https://cwe.mitre.org/data/definitions/543.html

③ The global statement, Python Software Foundation, https://docs.python.org/3/reference/simple_stmts.html#global

2. 제거되지 않고 남은 디버그 코드

가. 개요



디버깅 목적으로 삽입된 코드는 개발이 완료되면 제거해야 한다. 디버그 코드는 설정 등의 민감한 정보를 담거나 시스템을 제어하게 허용하는 부분을 담고 있을 수 있다. 만일, 남겨진 채로 배포될 경우, 공격자가 식별 과정을 우회하거나 의도하지 않은 정보와 제어 정보가 노출될 수 있다.

Django 프레임워크, Flask 프레임워크는 전역적으로 DEBUG 모드를 설정할 수 있다. DEBUG 모드를 사용하면 브라우저에서 임의의 Python 코드를 실행할 수도 있고 Python에서 발생한 모든 오류가 출력되어 정보노출의 위험이 있다. 어플리케이션을 배포 전에 반드시 DEBUG 모드를 비활성화 해야 한다.

나. 안전한 코딩기법

소프트웨어 배포 전, 반드시 디버그 코드를 확인 및 삭제한다. Django 프레임워크의 경우 전역적으로 DEBUG 모드를 비 활성화 하려면 settings.py 파일에 설정을 하고 Flask 프레임워크는 app_run() 전에 debug = False로 설정한다.

다. 코드예제

다음은 Django의 미들웨어 세팅 파일인 settings.py 파일이다. 개발 시 사용된 DEBUG 옵션이 True로 설정되어 있다.

가) Django 예제

안전하지 않은 코드의 예

```
1:  from django.urls import reverse_lazy
2:  from django.utils.text import format_lazy
3:
4:  DEBUG = True
5:
6:  ROOT_URLCONF = 'test.urls'
7:  SITE_ID = 1
8:
9:  DATABASES = {
10:     'default': {
11:         'ENGINE': 'django.db.backends.sqlite3',
12:         'NAME': ':memory:',
13:     }
14: }
```

개발이 끝난 소스코드를 운영, 배포 할 경우에는 DEBUG 옵션을 False로 변경해야 한다.

안전한 코드의 예

```
1:  from django.urls import reverse_lazy
2:  from django.utils.text import format_lazy
3:
4:  DEBUG = False
5:
6:  ROOT_URLCONF = 'test.urls'
7:  SITE_ID = 1
8:
9:  DATABASES = {
10:     'default': {
11:         'ENGINE': 'django.db.backends.sqlite3',
12:         'NAME': ':memory:',
13:     }
14: }
```

나) Flask 예제

다음은 Flask의 예제로 debug 모드가 True로 설정되어 있다.

안전하지 않은 코드의 예

```
1:  from flask import Flask
2:
3:  app = Flask(__name__)
4:  app.debug = True
5:
6:  @app.route('/')
7:  def hello_world():
8:      return 'Hello World!'
9:
10: if __name__ == '__main__':
11:     app.run()
```

개발이 끝난 소스코드를 운영, 배포 할 경우에는 debug 옵션을 False로 변경해야 한다.

안전한 코드의 예

```
1:  from flask import Flask
2:
3:  app = Flask(__name__)
4:  app.debug = False
5:
6:  @app.route('/')
7:  def hello_world():
8:      return 'Hello World!'
9:
10: if __name__ == '__main__':
11:     app.run()
```

라. 참고자료

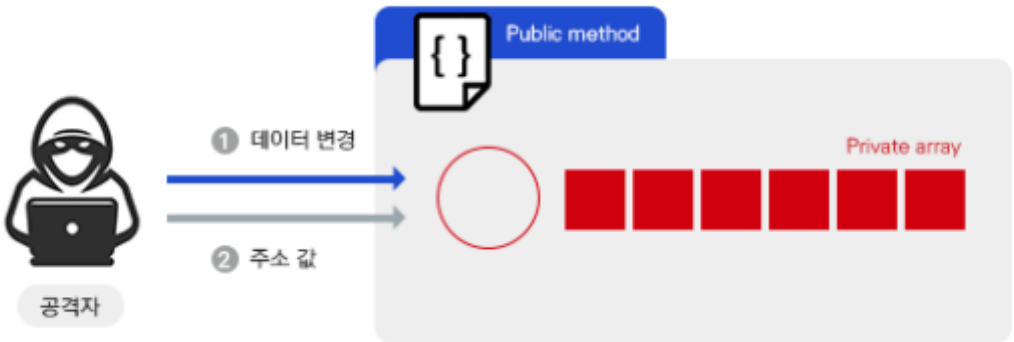
① CWE-489: Active Debug Code, MITRE,
https://cwe.mitre.org/data/definitions/489.html

② Settings, Django Software Foundation,
https://docs.djangoproject.com/en/3.2/ref/settings/#debug

③ Debug Mode, Flask,
https://flask.palletsprojects.com/en/2.0.x/quickstart/#debug-mode

3. Public 메소드로부터 반환된 Private 배열

가. 개요



Python은 명시적인 private 선언이 없다. 하지만 대부분의 Python 코드가 따르는 규칙으로 이름 앞에 밑줄(예:__spam)로 시작하면 Private으로 처리된다. 배열을 public으로 선언된 메소드를 통해 반환(return)하면, 그 배열의 레퍼런스가 외부에 공개되어 외부에서 배열 수정과 객체 속성변경이 가능해진다. 배열 뿐 아니라 변경 가능한(mutable) 객체에 대해 모두 발생한다.

구분	표시 방법
public	attribute, method는 기본적으로 public
protect	attribute, method 앞에 _(single underscore)를 붙여서 표시 함. 실제 제약 보다는 관례적임.
private	attribute, method 앞에 __(double underscore)를 붙여서 표시 함. 파이썬은 네임 맹글링(name mangling)으로 private 멤버에 _class__member로 접근은 가능하지만 바람직하지 않음.

나. 안전한 코딩기법

private로 선언된 배열을 public으로 선언된 메소드를 통해 반환하지 않도록 해야 한다. private 배열에 대한 복사본을 반환하도록 하고 배열의 원소에 대해서는 clone() 메소드를 통해 복사된 원소를 저장하도록 하여 private 선언된 배열과 객체 속성에 대한 의도하지 않게 수정되는 것을 방지한다. 만약 배열의 원소가 String 타입 등과 같이 변경이 되지 않는 경우에는 Private 배열의 복사본을 만들고 이를 반환하도록 작성한다.

다. 코드예제

다음 예제는 __를 이용해서 Python의 private 배열을 생성하고 그것을 반환하는 public 메소드를 사용하고 있다. 이 경우 특정 배열 타입에 따라 외부에서 private 배열을 변조할 수 있는 문제를 내포하고 있다.

안전하지 않은 코드의 예

```
1:  import copy
2:  class UserObj():
3:      __private_variable = []
4:      def __init__(self):
5:          pass
6:
7:      # private 배열을 리턴하는 public 메소드를 사용하는 경우 취약함
8:      def get_private_member(self):
9:          return self.__private_variable
```

아래 예제는 내부와 외부의 배열이 서로 참조되는 것을 예방하기 위해 [:]로 새로운 객체를 생성하여 값을 반환하고 있다.

안전한 코드의 예

```
1:  class UserObj():
2:      __private_variable = []
3:      def __init__(self):
4:          pass
5:
6:      # private 배열을 반환하는 경우 [:]를 사용하여 외부와 내부의
7:      # 배열이 서로 참조되지 않도록 해야 한다
8:      def get_private_member(self):
9:          return self.__private_variable[:]
```

라. 참고자료

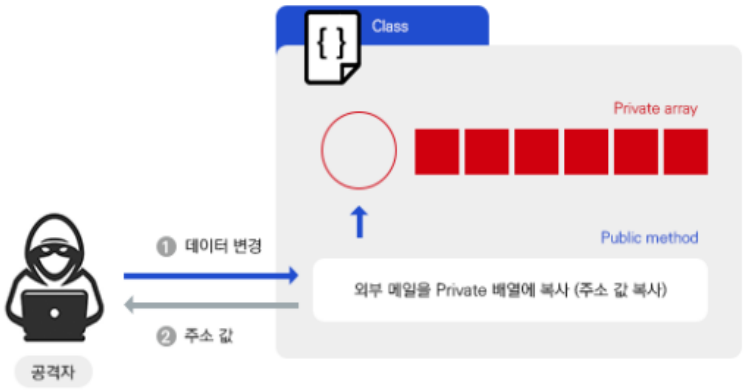
① CWE-495: Private Data Structure Returned From A Public Method, MITRE, https://cwe.mitre.org/data/definitions/495.html

② Do not return references to private mutable class members, CERT, https://wiki.sei.cmu.edu/confluence/display/java/OBJ05-J.+Do+not+return+references+to+private+mutable+class+members

③ Shallow and deep copy operations, Python Software Foundation, https://docs.python.org/3/library/copy.html

4. Private 배열에 Public 데이터 할당

가. 개요



public으로 선언된 메소드의 인자가 private 선언된 배열에 저장되면, private 배열을 외부에서 접근하여 배열 수정과 객체 속성 변경이 가능해진다.

나. 안전한 코딩기법

public으로 선언된 메소드의 인자를 private선언된 배열로 저장되지 않도록 한다. 인자로 들어온 배열의 복사본을 생성하고 clone() 메소드를 통해 복사된 원소를 저장하도록 하여 private 배열에 할당하여 private 선언된 배열과 객체속성에 대한 의도하지 않게 수정되는 것을 방지한다. 만약 배열 객체의 원소가 String 타입 등과 같이 변경이 되지 않는 경우에는 인자로 들어온 배열의 복사본을 생성하여 할당한다.

다. 코드예제

다음 예제는 __를 이용해서 Python의 내부 배열을 생성하고 외부 값을 대입하는 public 메소드를 사용하고 있다. 이 경우 특정 배열 타입에 따라 외부에서 private 배열을 변조할 수 있는 문제를 내포하고 있다.

안전하지 않은 코드의 예

```
1:  import copy
2:  class UserObj():
3:      __private_variable = []
4:      def __init__(self):
5:          pass

6:      # private 배열에 외부 값을 바로 대입하는 public 메소드를 사용하는
7:      # 경우 취약함
8:      def set_private_member(self, num):
9:          self.__private_variable = num
```

아래 예제는 내부와 외부의 배열이 서로 참조되는 것을 예방하기 위해 [:]로 새로운 객체를 생성하여 값을 대입하고 있다.

안전한 코드의 예

```
1:  class UserObj():
2:      def __init__(self):
3:          self.__privateVariable = []
4:
5:      # private 배열에 외부 값을 바로 대입하는 경우 [:]를 사용하여
6:      # 외부와 내부의 배열이 서로 참조되지 않도록 해야 함
7:      def set_private_member(self, num):
8:          self.__privateVariable = num[:]
```

라. 참고자료

① CWE-496: Public Data Assigned to Private Array-Typed Field, MITRE, https://cwe.mitre.org/data/definitions/496.html

② Shallow and deep copy operations, Python Software Foundation, https://docs.python.org/3/library/copy.html

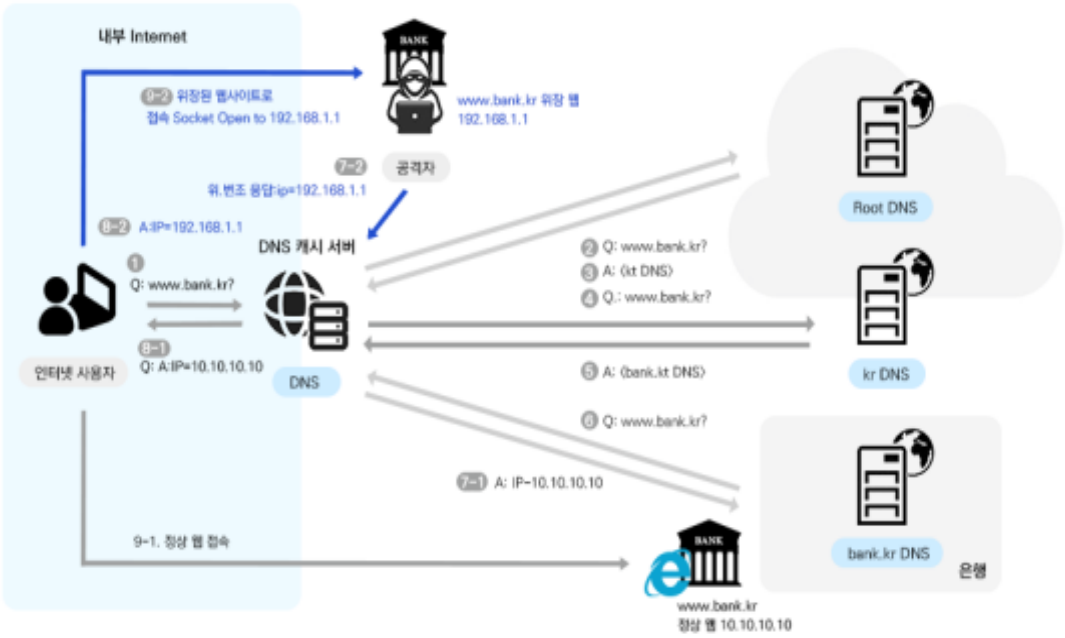
③ Private Variables, Python Software Foundation, https://docs.python.org/3/tutorial/classes.html#private-variables

제7절 API 오용

의도된 사용에 반하는 방법으로 API를 사용하거나, 보안에 취약한 API를 사용하여 발생할 수 있는 보안약점이다.

1. DNS lookup에 의존한 보안결정

가. 개요



공격자가 DNS 엔트리를 속일 수 있으므로 도메인명에 의존에서 보안결정(인증 및 접근 통제 등)을 하지 않아야 한다. 만약, 로컬 DNS 서버의 캐시가 공격자에 의해 오염된 상황이라면, 사용자와 특정 서버 간의 네트워크 트래픽이 공격자를 경유하도록 할 수도 있다. 또한, 공격자가 마치 동일 도메인에 속한 서버인 것처럼 위장할 수도 있다.

나. 안전한 코딩기법

보안결정에서 도메인명을 이용한 DNS lookup을 하지 않도록 한다.

다. 코드예제

다음의 예제는 도메인명을 통해 해당 요청을 신뢰할 수 있는지를 검사한다. 그러나 공격자는 DNS 캐쉬 등을 조작해서 쉽게 이러한 보안 설정을 우회할 수 있다.

안전하지 않은 코드의 예

```
1:  import socket
2:
3:  def is_trust(host_domain_name):
4:      trusted = False
5:      trusted_host = "trust.example.com"
6:      # 공격자에 의해 실행되는 서버의 DNS가 변경될 수 있으므로
7:      # 안전하지 않다
        if trusted_host == host_name:
8:          trusted = True
9:      return trusted
```

도메인명을 이용한 비교를 하지 말고 IP 주소를 직접 비교하도록 수정한다.

안전한 코드의 예

```
1:  import socket
2:
3:  def is_trust(host_domain_name):
4:      trusted = False
5:      trusted_ip = "192.168.10.7"
6:      # 실제 서버의 IP 주소를 비교하여 DNS 변조에 대응
7:      dns_resolved_ip = socket.gethostbyaddr(host_domain_name)
8:
9:      if trusted_ip == dns_resolved_ip:
10:         trusted = True
11:     return trusted
```

라. 참고자료

① CWE-350: Reliance on Reverse DNS Resolution for a Security-Critical Action, MITRE, https://cwe.mitre.org/data/definitions/350.html

② Socket, Python Software Foundation, https://docs.python.org/3/library/socket.html

3

PART 제3장

부 록

제1절 구현단계 보안약점 제거 기준

제2절 용어정리

3 부록

제1절 구현단계 보안약점 제거 기준

1. 입력데이터 검증 및 표현

번호	보안약점	설명
1	SQL 삽입	SQL 질의문을 생성할 때 검증되지 않은 외부 입력값을 허용하여 악의적인 질의문이 실행가능한 보안약점
2	코드 삽입	프로세스가 외부 입력값을 코드(명령어)로 해석·실행할 수 있고 프로세스에 검증되지 않은 외부 입력값을 허용한 경우 악의적인 코드가 실행 가능한 보안약점
3	경로 조작 및 자원 삽입	시스템 자원 접근경로 또는 자원제어 명령어에 검증되지 않은 외부 입력값을 허용하여 시스템 자원에 무단 접근 및 악의적인 행위가 가능한 보안약점
4	크로스사이트 스크립트	사용자 브라우저에 검증되지 않은 외부 입력값을 허용하여 악의적인 스크립트가 실행 가능한 보안약점
5	운영체제 명령어 삽입	운영체제 명령어를 생성할 때 검증되지 않은 외부 입력값을 허용하여 악의적인 명령어가 실행 가능한 보안약점
6	위험한 형식 파일 업로드	파일의 확장자 등 파일형식에 대한 검증없이 파일 업로드를 허용하여 공격이 가능한 보안약점
7	신뢰되지 않는 URL 주소로 자동접속 연결	URL 링크 생성에 검증되지 않은 외부 입력값을 허용하여 악의적인 사이트로 자동 접속 가능한 보안약점
8	부적절한 XML 외부 개체 참조	임의로 조작된 XML 외부개체에 대한 적절한 검증 없이 참조를 허용하여 공격이 가능한 보안약점
9	XML 삽입	XQuery, XPath 질의문을 생성할 때 검증되지 않은 외부 입력값을 허용하여 악의적인 질의문이 실행 가능한 보안약점
10	LDAP 삽입	LDAP 명령문을 생성할 때 검증되지 않은 외부 입력값을 허용하여 악의적인 명령어가 실행 가능한 보안약점
11	크로스사이트 요청 위조	사용자 브라우저에 검증되지 않은 외부 입력값을 허용하여 사용자 본인의 의지와는 무관하게 공격자가 의도한 행위가 실행 가능한 보안약점
12	서버사이드 요청 위조	서버 간 처리되는 요청에 검증되지 않은 외부 입력값을 허용하여 공격자가 의도한 서버로 전송하거나 변조하는 보안약점
13	HTTP 응답분할	HTTP 응답헤더에 개행문자(CR이나 LF)가 포함된 검증되지 않은 외부 입력값을 허용하여 악의적인 코드가 실행 가능한 보안약점
14	정수형 오버플로우	정수형 변수에 저장된 값이 허용된 정수 값 범위를 벗어나 프로그램이 예기치 않게 동작 가능한 보안약점

번호	보안약점	설명
15	보안기능 결정에 사용되는 부적절한 입력값	보안기능(인증, 권한부여 등) 결정에 검증되지 않은 외부 입력값을 허용하여 보안기능을 우회하는 보안약점
16	메모리 버퍼 오버플로우	메모리 버퍼의 경계값을 넘어서 메모리값을 읽거나 저장하여 예기치 않은 결과가 발생하는 보안약점
17	포맷 스트링 삽입	str.format등 포맷 스트링 제어함수에 검증되지 않은 외부 입력값을 허용하여 발생하는 보안약점 * 포맷 스트링: 입·출력에서 형식이나 형태를 지정해주는 문자열

2. 보안기능

번호	보안약점	설명
1	적절한 인증 없는 중요 기능 허용	중요정보(금융정보, 개인정보, 인증정보 등)를 적절한 인증없이 열람(또는 변경) 가능한 보안약점
2	부적절한 인가	중요자원에 접근할 때 적절한 제어가 없어 비인가자의 접근이 가능한 보안 약점
3	중요한 자원에 대한 잘못된 권한 설정	중요자원에 적절한 접근 권한을 부여하지 않아 중요정보가 노출·수정 가능한 보안약점
4	취약한 암호화 알고리즘 사용	중요정보 (금융정보, 개인정보, 인증정보 등)의 기밀성을 보장할 수 없는 취약한 암호화 알고리즘을 사용하여 정보가 노출 가능한 보안 약점
5	암호화되지 않은 중요정보	중요정보(비밀번호, 개인정보 등) 전송 시 암호화 또는 안전한 통신채널을 이용하지 않거나, 저장 시 암호화 하지 않아 정보가 노출 가능한 보안약점
6	하드코드된 중요정보	소스코드에 중요정보(비밀번호, 암호화키 등)를 직접 코딩하여 소스코드 유출 시 중요정보가 노출되고 주기적 변경이 어려운 보안약점
7	충분하지 않은 키 길이 사용	암호화 등에 사용되는 키의 길이가 충분하지 않아 데이터의 기밀성·무결성을 보장할 수 없는 보안약점
8	적절하지 않은 난수 값 사용	사용한 난수가 예측 가능하여, 공격자가 다음 난수를 예상해서 시스템을 공격 가능한 보안약점
9	취약한 비밀번호 허용	비밀번호 조합규칙(영문, 숫자, 특수문자 등) 미흡 및 길이가 충분하지 않아 비밀번호가 노출 가능한 보안약점
10	부적절한 전자서명 확인	프로그램, 라이브러리, 코드의 전자서명에 대한 유효성 검증이 적절하지 않아 공격자의 악의적인 코드가 실행 가능한 보안약점
11	부적절한 인증서 유효성 검증	인증서에 대한 유효성 검증이 적절하지 않아 발생하는 보안약점
12	사용자 하드디스크에 저장되는 쿠키를 통한 정보노출	쿠키(세션 ID, 사용자 권한정보 등 중요정보)를 사용자 하드디스크에 저장하여 중요정보가 노출 가능한 보안약점
13	주석문 안에 포함된 시스템 주요정보	소스코드 주석문에 인증정보 등 시스템 주요정보가 포함되어 소스코드 노출 시 주요정보도 노출 가능한 보안약점
14	솔트 없이 일방향 해쉬 함수 사용	솔트를 사용하지 않고 생성된 해쉬 값으로부터 공격자가 미리 계산된 레인보우 테이블을 이용하여 해쉬 적용 이전 원본 정보를 복원가능한 보안약점 *솔트: 해쉬 적용하기 전 평문인 전송정보에 덧붙인 무의미한 데이터
15	무결성 검사 없는 코드 다운로드	소스코드 또는 실행파일을 무결성 검사 없이 다운로드 받아 실행하는 경우, 공격자의 악의적인 코드가 실행 가능한 보안약점
16	반복된 인증시도 제한 기능 부재	인증 시도 수를 제한하지 않아 공격자가 반복적으로 임의 값을 입력하여 계정 권한을 획득 가능한 보안약점

3. 시간 및 상태

번호	보안약점	설명
1	경쟁조건 : 검사 시점과 사용 시점	멀티 프로세스 상에서 자원을 검사하는 시점과 사용하는 시점이 달라서 발생하는 보안약점
2	종료되지 않는 반복문 또는 재귀함수	종료조건 없는 제어문 사용으로 반복문 또는 재귀함수가 무한히 반복되어 발생할 수 있는 보안약점

4. 에러처리

번호	보안약점	설명
1	오류 메시지 정보노출	오류메시지나 스택정보에 시스템 내부구조가 포함되어 민감한 정보, 디버깅 정보가 노출 가능한 보안약점
2	오류상황 대응 부재	시스템 오류상황을 처리하지 않아 프로그램 실행정지 등 의도하지 않은 상황이 발생 가능한 보안약점
3	부적절한 예외처리	예외사항을 부적절하게 처리하여 의도하지 않은 상황이 발생 가능한 보안약점

5. 코드오류

번호	보안약점	설명
1	Null Pointer 역참조	변수의 주소 값이 Null인 객체를 참조하는 보안약점
2	부적절한 자원 해제	사용 완료된 자원을 해제하지 않아 자원이 고갈되어 새로운 입력을 처리할 수 없는 보안약점
3	해제된 자원 사용	메모리 등 해제된 자원을 참조하여 예기치 않은 오류가 발생하는 보안약점
4	초기화되지 않은 변수 사용	변수를 초기화하지 않고 사용하여 예기치 않은 오류가 발생하는 보안약점
5	신뢰할 수 없는 데이터의 역직렬화	악의적인 코드가 삽입·수정된 직렬화 데이터를 적절한 검증 없이 역직렬화하여 발생하는 보안약점 * 직렬화: 객체를 전송 가능한 데이터형식으로 변환 * 역직렬화: 직렬화된 데이터를 원래 객체로 복원

6. 캡슐화

번호	보안약점	설명
1	잘못된 세션에 의한 데이터 정보노출	잘못된 세션에 의해 인가되지 않은 사용자에게 중요정보가 노출 가능한 보안약점
2	제거되지 않고 남은 디버그 코드	디버깅을 위한 코드를 제거하지 않아 인가되지 않은 사용자에게 중요정보가 노출 가능한 보안약점
3	Public 메서드로부터 반환된 Private 배열	Public으로 선언된 메소드에서 Private로 선언된 배열을 반환(return)하면 Private 배열의 주소값이 외부에 노출되어 해당 Private 배열값을 외부에서 수정 가능한 보안약점
4	Private 배열에 Public 데이터 할당	Public으로 선언된 데이터 또는 메소드의 인자가 Private으로 선언된 배열에 저장되면 Private 배열을 외부에서 접근하여 수정 가능한 보안약점

7. API 오용

번호	보안약점	설명
1	DNS lookup에 의존한 보안결정	도메인명 확인(DNS lookup)으로 보안결정을 수행할 때 악의적으로 변조된 DNS 정보로 예기치 않은 보안위협에 노출되는 보안약점
2	취약한 API 사용	취약한 함수를 사용해서 예기치 않은 보안위협에 노출되는 보안약점

제2절 용어정리

● Developer Economics State of the Developer Nation, 20th Edition

developernation.net에서 매년 165개국 30,000명 이상의 개발자들을 대상으로 설문조사를 하여 제공하고 있다. 웹, 모바일, 데스크톱, 클라우드, 산업용 IoT, 소비자 전자 제품, 임베디디소프트웨어, AR 및 VR등 다양한 분야의 설문을 실시하고 있다.

● Developer AES(Advanced Encryption Standard)

미국 정부 표준으로 지정된 블록 암호 형식으로 이전의 DES를 대체하며, 미국 표준 기술 연구소 (NIST)가 5년의 표준화 과정을 거쳐 2001년 11월 26일에 연방 정보처리표준(FIPS 197)으로 발표하였다.

● DES 알고리즘

DES(Data Encryption Standard)암호는 암호화 키와 복호화키가 같은 대칭키 암호로 64비트의 암호화키를 사용한다. 전수공격(Brute Force)공격에 취약하다.

● HMAC(Hash-based Message Authentication Code)

해쉬 기반 메시지 인증 코드, 메시지 다이제스트 알고리듬 5(MD-5), SHA-1 등 반복적인 암호화 해쉬 기능을 비밀 공용키와 함께 사용하며, 체크섬을 변경하는 것이 불가능하도록 한 키 기반의 메시지 인증 알고리즘이다.

● HTTPS(Hypertext Transfer Protocol over Secure Socket Layer)

WWW(월드 와이드 웹) 통신 프로토콜인 HTTP의 보안이 강화된 버전이다.

● LDAP(Lightweight Directory Access Protocol)

TCP/IP 위에서 디렉토리 서비스를 조회하고 수정하는 응용 프로토콜이다.

● SHA(Secure Hash Algorithm)

해쉬알고리즘의 일종으로 MD5의 취약성을 대신하여 사용한다. SHA, SHA-1, SHA-2(SHA-224, SHA-256, SHA-384, SHA-512) 등의 다양한 버전이 있으며, 암호 프로토콜인 TLS, SSL, PGP, SSH, IPSec 등에 사용된다.

● umask

파일 또는 디렉토리의 권한을 설정하기 위한 명령어이다.

● 개인키(Private Key)

공개키 기반구조에서 개인키란 암·복호화를 위해 비밀 메시지를 교환하는 당사자만이 알고 있는 키이다.

● 공개키(Public Key)

공개키는 지정된 인증기관에 의해 제공되는 키값으로서, 이 공개키로부터 생성된 개인키와 함께 결합되어, 메시지 및 전자서명의 암·복호화에 효과적으로 사용될 수 있다. 공개키를 사용하는 시스템을 공개키 기반구조(Public Key Infrastructure, PKI)라 한다.

● 경로순회(directory traversal)

상대경로 참조 방식(“./”,“../”등)을 이용해 다른 디렉토리의 중요파일에 접근하는 공격방법으로 경로 추적이라고도 한다.

● 동적 SQL(Dynamic SQL)

프로그램의 조건에 따라 SQL문이 다르게 생성되는 경우, 프로그램 실행 시에 전체 쿼리문이 완성되어 DB에 요청하는 SQL문을 말한다.

● 동적 쿼리(Dynamic Query)

컬럼이나 테이블명을 바꿔 SQL 쿼리를 실시간 생성해 DB에 전달하여 처리하는 방식이다.

● 소프트웨어 개발보안

소프트웨어 개발과정에서 개발자 실수, 논리적 오류 등으로 인해 소프트웨어에 내재된 보안취약점 을 최소화하는 한편, 해킹 등 보안위협에 대응할 수 있는 안전한 소프트웨어를 개발하기 위한 일련 의 과정을 의미한다. 넓은 의미에서 소프트웨어 개발보안은 소프트웨어 생명주기의 각 단계별로 요구되는 보안활동을 모두 포함하며, 좁은 의미로는 SW개발과정에서 소스코드를 작성하는 구현 단계에서 보안취약점을 배제하기 위한 '시큐어코딩(Secure Coding)'을 의미한다.

● 소프트웨어 보안약점

소프트웨어 결함, 오류 등으로 해킹 등 사이버공격을 유발할 가능성이 있는 잠재적인 보안취약점을 말한다.

● 싱글톤 패턴(Singleton Pattern)

하나의 프로그램 내에서 하나의 인스턴스만을 생성해야만 하는 패턴이다. Connection Pool, Thread Pool과 같이 Pool 형태로 관리되는 클래스의 경우 프로그램 내에서 단 하나의 인스턴트로 관리해야 하는 경우를 말함. Python에서는 객체로 제공된다.

● 정적 쿼리(Static Query)

동적 쿼리와 달리 프로그램 소스코드에 이미 쿼리문이 완성된 형태로 고정되어 있다.

● 해쉬함수

주어진 원문에서 고정된 길이의 의사난수를 생성하는 연산기법이며, 생성된 값은 '해쉬값'이라고 한다. MD5, SHA, SHA-1, SHA-256 등의 알고리즘이 있다.

● 화이트 리스트(White List)

블랙리스트(Black List)의 반대개념으로 신뢰할 수 있는 사이트나 IP주소 목록을 말한다.

● 장고 웹 프레임워크(Django Web Framework)

파이썬으로 작성된 오픈 소스 웹 프레임워크로, 모델(Model)-뷰(View)-컨트롤러(Controller)의 MVC패턴을 따르고 있다. 전통적인 MVC 디자인 패턴에서 이야기하는 컨트롤러의 기능을 프레임워크 자체에서 처리하기에 모델(Model), 템플릿(Template), 뷰(View)로 분류해 MTV 프레임워크라고 하기도 한다. 컴포넌트 재사용성과 플러그인화 가능성, 빠른 개발 등을 강조하고 있다.

● 플라스크 웹 프레임워크(Flask Web Framework)

파이썬으로 작성된 마이크로 웹 프레임워크의 하나이며, 특별한 도구나 라이브러리가 필요 없기 때문에 마이크로 프레임워크라고 부른다.

● 파싱(Parsing)

일련의 문자열을 의미 있는 token(어휘 분석의 단위)으로 분해하고 그것들로 이루어진 Parse tree를 만드는 과정이다. 어떤 문장을 분석하거나 문법적 관계를 해석하는 행위를 말한다.

● 파서(Parser)

컴파일러(compiler)의 일부로 컴파일러나 인터프리터(Interpreter)에서 원시 프로그램을 읽어 들여 그 문장의 구조를 알아내는 parsing(구문 분석)을 행하는 프로그램을 말한다.

● XML(eXtensible Markup Language)

W3C에서 개발되었으며, 다른 특수한 목적을 갖는 마크업 언어를 만드는데 사용된다. 인터넷에 연결된 시스템끼리 데이터를 쉽게 주고받을 수 있어 HTML의 한계를 극복할 목적으로 만들어졌다.

● DTD(Document Type Definition)

문서 타입 정의(DTD)는 XML 문서의 구조 및 해당 문서에서 사용할 수 있는 적법한 요소와 속성을 정의한다.

● Decorator

함수를 받아 명령을 추가한 뒤 이를 다시 함수의 형태로 반환하는 함수이다. 반복을 줄이고 메소드나 함수의 책임을 확장할 수 있으며 재사용이 가능하게 해준다. 파이썬에서 @로 시작하는 구문으로 표시한다.

● 공개 키 인증서(Public Key Certificate)

공개키의 소유권을 증명하는데 사용되는 전자 문서이다. 키에대한 정보, 소유자의 신원에 대한 정보, 발급자의 디지털 서명이 포함되어 있다.