소스코드 보안취약점 자동 탐지를 위한 취약점 패턴 명세 언어

김현하, 도경구 :: 한양대학교 프로그래밍언어연구실
{hhkim, doh} @ plasse.hanyang.ac.kr

제4회 소프트웨어무결점연구센터 Workshop
2010.8.25~28 @ 대명 리조트 설악

Vulnerability Pattern Description Language

소스코드 취약점 패턴 명세 언어

- 프로그램에서 찾으려는 소스코드 보안 취약점 패턴을 기술하기 위한 언어
  - 변환된 중간언어에서 작성된 취약점 패턴들을 자동으로 검출
- 하나의 취약점패턴은,
  - 코드의 특정한 형태
  - 여러 지점간의 관계 = 제어 흐름, 데이터 의존성
- 패턴분석기는 "코드의 특정한 형태"에 해당하는 지점(+부가정보)을 찾아냄

흐름패턴 $p_1 \rightarrow p_2 \rightarrow \ldots \rightarrow p_n$ 에서 각 패턴들과 $\{p_1, p_2, \ldots, p_n\}$

단일패턴 p_0 의 패턴을 $\{p_0\}$

하나로 모아서 $\{p_0, p_1, p_2, \ldots, p_n\}$

$p \rightarrow position \times information$ 각 패턴이 어떤 위치(+부가정보) 찾아졌는지

$position \rightarrow 2^p \times information$ 각 위치에서 어떤 패턴(+부가정보)이 찾아졌는지

를 구해준다.

여기서 부가정보는 패턴의 유형에 따라 함수의이름, 식별자의위치 등을 포함한다.

예제

[ServerSocket(...)]

serverSocket 함수의 호출

```
public void ejb_function() throws IOException {
    // ...
      ServerSocket s = new ServerSocket( );
    // ...
}
```

[DEF(clone)] in-subclass-of Cloneable

Cloneable의 subclass 에서 clone함수의 선언

```
public class R31000491 implements Cloneable {
  // ...
  protected Object clone() throws
      CloneNotSupportedException {
    // ...
  }
}
```

[getParameter(...)] -->df [setAttribute(..., $0)]

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
  usrname = request.getParameter("usrname");
  if (session.getAttribute(ATTR_USR) == null){
    session.setAttribute(ATTR_USR, usrname);
  }
}
```

getParameter, setAttribute 메소드 호출

- 흐름분석기는 찾아낸 "지점간의 관계"를 알아냄
  - 흐름의 종류에 대한 구분
    - 제어 흐름 : control flow (CF)
    - 직접 자료 흐름 : direct data flow (DF)
      - 값의 변형 없이 바로 전달 되는 경우
    - 간접 자료 흐름 : indirect data flow (IF)
      - 값이 함수나 연산을 통해서 변경되어 전달
  - 흐름의 한정자에 대한 구분
    - 존재 한정 흐름

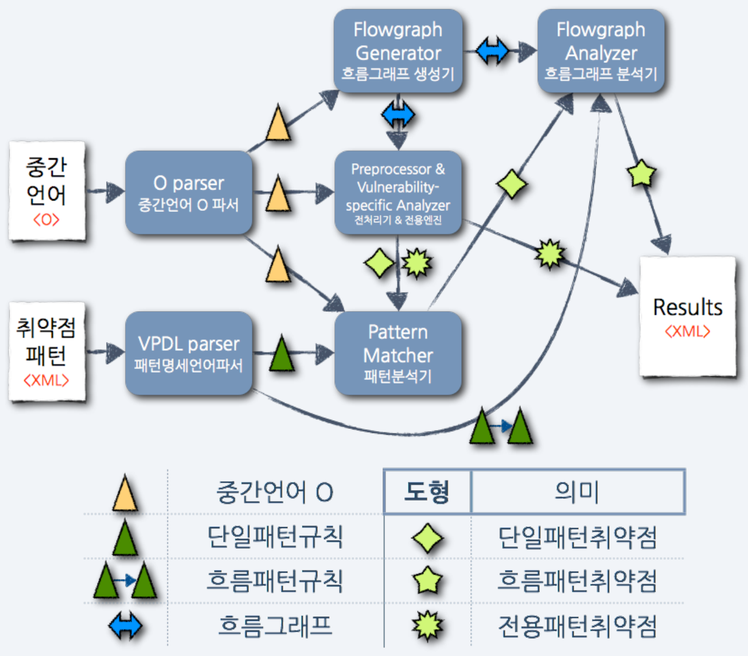
      p_1 --> p_2 : p_1 이후에 존재하는 모든 경로 중 하나 이상에서 p_2 가 존재
    - 전체 한정 흐름

      p_1 ==> p_2 : p_1 이후에 존재하는 모든 경로에서 p_2가 존재
  - 흐름의 방향에 대한 구분
    - 정방향 흐름

      p_1 --> p_2 : 이후에 존재하는 모든 경로 중 하나 이상에서 p_2 가 존재하는 p_1
    - 역방향 흐름

      p_1 <-- p_2 : 이전에 존재하는 모든 경로 중 하나 이상에서 p_1 가 존재하는 p_2
    - 흐름의 방향과 상관없이 프로그램의 순서는 좌측에서 우측으로 진행

소스코드 보안 취약점 분석기 전체그림



취약점 패턴 분석 예제

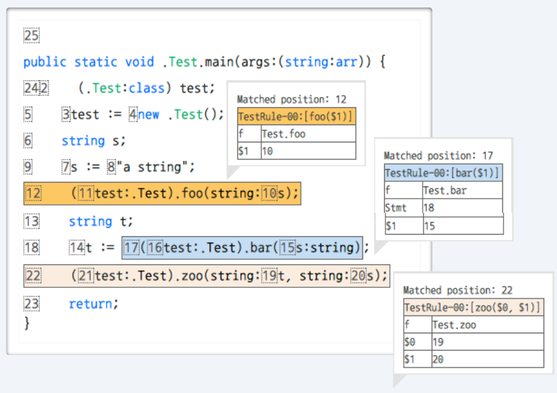
예제 코드

```
public class Test {

   void foo(String s) { ... }

   String bar(String s) {
      ...
      return s;
   }

   void zoo(String s,String t) { ... }


   public static void main(String[] args) {
     Test test = new Test();

     String s = "a string";

     test.foo(s);
     String t = test.bar(s);
     test.zoo(t, s);
   }
}
```

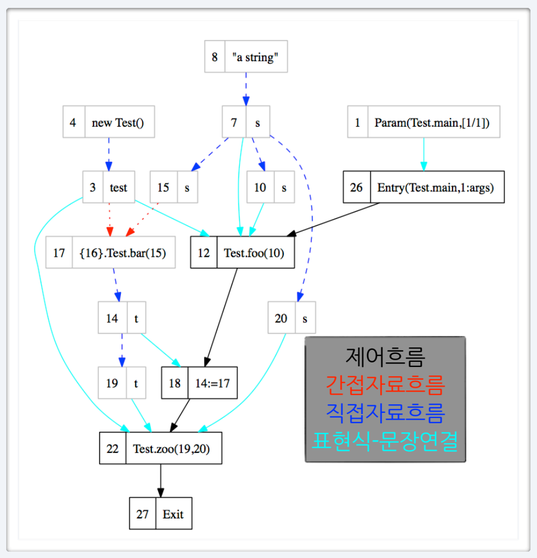
예제 취약점 패턴

```
<RuleSet>
  <Rule RuleID="TestRule-00">
    <CheckRule>
      <UNSAFE>
        [foo($1)]
          -->df [bar($1)]
          -->df [zoo($0,$1)]
      </UNSAFE>
    <CheckRule>
  </Rule>
</RuleSet>
```

패턴분석기 결과



흐름그래프



흐름분석 결과 로그

```
curr:[foo($1)]
  [TestRule-00]
  find nexts of 12 ...
  17 is ep, use alternative  sp : 18   [12 df-> 18]  -----
  12's nexts = {18:CF}
   |--> {17}
    next:[bar($1)]/17
    **** chk bounding ... ********************
    $1 :  [10 df-> 15]  matched!
    ************************************** 0
    ************* goto next step **************
17 is ep, use alternative  sp : 18    [TestRule-00]
    find nexts of 18 ...
       [18 df-> 22]  -----
    18's nexts = {22:CF}
     |--> {22}
      next:[zoo($0, $1)]/22
      **** chk bounding ... ********************
      $1 :  [15 df-> 20]  matched!
      $0 :  [17 df-> 19]  matched!
      ************************************** 0
      ****************** hit! *******************
```

취약점 분석결과

```
<AnalysisResults xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
xsi:noNamespaceSchemaLocation='resultset.xsd'>
 <Target Filename='Test.java' RelativePath='./'>
  <Position RuleID='TestRule-00' From='19' To='19'/>
 </Target>
</AnalysisResults>
```

소스코드 보안취약점 자동 탐지를 위한 취약점 패턴 명세 언어

김현하, 도경구 :: 한양대학교 프로그래밍언어연구실
{hhkim, doh} @ plasse.hanyang.ac.kr

ROSAEC center
Research On Software Analysis for Error-free Computing
소프트웨어 무결점 연구센터 KOSEF ERC
제4회 소프트웨어무결점연구센터 Workshop
2010.8.25~28 @ 대명 리조트 설악

취약점 패턴 명세 언어의 문법구조 확장BNF 형식

취약점 패턴

vulnerability-pattern	::=	simple-pattern	단일 패턴
	|	flow-pattern	흐름 패턴
	|	XOR [simple-pattern ; simple-pattern]	논리연산자 XOR
	|	AND [simple-pattern | flow-pattern { ; simple-pattern | flow-pattern }]	논리연산자 AND
	|	VSP [id , n]	전용분석
simple-pattern	::=	pattern	패턴
	|	CMT [*"literal"* { , *"literal"* }]	주석검사
flow-pattern	::=	pattern flow pattern	단일 흐름 패턴
	|	pattern flow flow-pattern	다중 흐름 패턴
flow	::=	arrow sort	흐름
arrow	::=	-->	정방향 존재 한정 흐름
	|	<--	역방향 존재 한정 흐름
	|	==>	정방향 전체 한정 흐름
	|	<==	역방향 전체 한정 흐름
sort	::=	cf	제어 흐름
	|	df	직접전달 자료 흐름
	|	if	간접전달 자료 흐름

선언부 패턴

def-pattern	::=	FIELD (rid) [qual]	필드변수 선언
	|	DEF (fid) [qual] [throws]	함수 / 메소드 선언
qual	::=	qualified-by qualifier { , qualifier }	한정자
throws	::=	throws exen { , exen }	예외처리
qualifier	::=	PUBLIC | PRIVATE | PROTECTED | PACKAGE | INTERNAL | DELEGATE	
	|	FINAL | EVENT | STATIC | ABSTRACT | TRANSIENT | PACKAGE	
	|	SYNCHRONIZED | STRICTFP | VOLATILE | NATIVE | EXTERN	
	|	OVERRIDE | NEW | VIRTUAL | IMPLICIT | EXPLICIT | UNSAFE	
	|	READONLY | SEALED	

문장부 패턴

stmt-pattern	::=	ASSIGN (rid := rid)	선언문
	|	throw (exen)	예외 발생
	|	return	return 문

표현식 패턴

expr-pattern	::=	funcall	함수호출
	|	PKG package	패키지 사용
	|	rid	RDL 식별자
	|	(rid , bop , rid)	이항연산
funcall	::=	fid ([param { , param }])	
param	::=	rid | ...	파라미터
bop	::=	== | != | > | >= | < | < | = | + | - | * | /	이항연산자

식별자

fid	::=	id { . id }	함수 / 메소드
	|	rid . id	인스턴스의 메소드
	|	$$	임의의 함수
	|	$ n	임의의 함수
package	::=	id	패키지
	|	id . package	
	|	id . *	
class	::=	id { . id }	클래스
name	::=	id	변수명
	|	class . id	멤버명
exen	::=	id { . id }	예외

어휘

n	::=	[0 - 9]*
id	::=	[a-zA-Z0-9_-]*
literal	::=	{ string_character }
string_character	::=	[^"]
	|	\"

패턴

pattern	::=	pat [in-expr] { in-stmt } { in-decl }	패턴
pat	::=	[def-pattern]	선언부 패턴
	|	[stmt-pattern]	문장부 패턴
	|	[expr-pattern]	표현식 패턴
	|	[region]	영역
in-expr	::=	in expr-region	표현식 영역 제한
	|	not-in expr-region	
in-stmt	::=	in stmt-region	문장부 영역 제한
	|	not-in stmt-region	
in-decl	::=	in decl-region	선언부 영역 제한
	|	not-in decl-region	
	|	in-function fid	
	|	not-in-function fid	
	|	in-class class	
	|	not-in-class class	
	|	in-subclass-of class	
	|	not-in-subclass-of class	

영역

region	::=	expr-region	표현식 영역
	|	stmt-region	문장부 영역
	|	decl-region	선언부 영역
expr-region	::=	IFCOND	조건문의 조건표현식
	|	LOOPCOND	반복문의 조건표현식
stmt-region	::=	TRY	
	|	CATCH (exen)	
	|	FINALLY	
	|	SYNCHRONIZED	
	|	IFBODY	
	|	LOOPBODY	
	|	EMPTY	
decl-region	::=	CONSTRUCTOR	

RDL 식별자

rid	::=	$ [n] [< type >] [< attr { , attr } >]	RDL 식별자
	|	$$	조건없는 RDL 식별자
	|	*"literal"* < type >	리터럴
	|	null	널 리터럴
type	::=	CLASS class	클래스
	|	ARRAY	배열
	|	INT	정수형. (unsigned) byte, (unsigned) short, (unsigned) int, long 포함
	|	FLOAT	실수형. float, double, decimal 포함
	|	CHAR | STRING | BOOL	
attr	::=	NULLABLE	널 값이 가능
	|	UNSIGNED	unsigned 류의 타입
	|	CONSTANT	상수
	|	POSITIVE	양의 정수
	|	NEGATIVE	음의 정수
	|	ZERO	0
	|	FIELD	필드 접근
	|	EMPTYSTRING	빈 문자열
	|	EXTERN	명령줄 인자
	|	NAME name	해당 이름을 가지는 변수