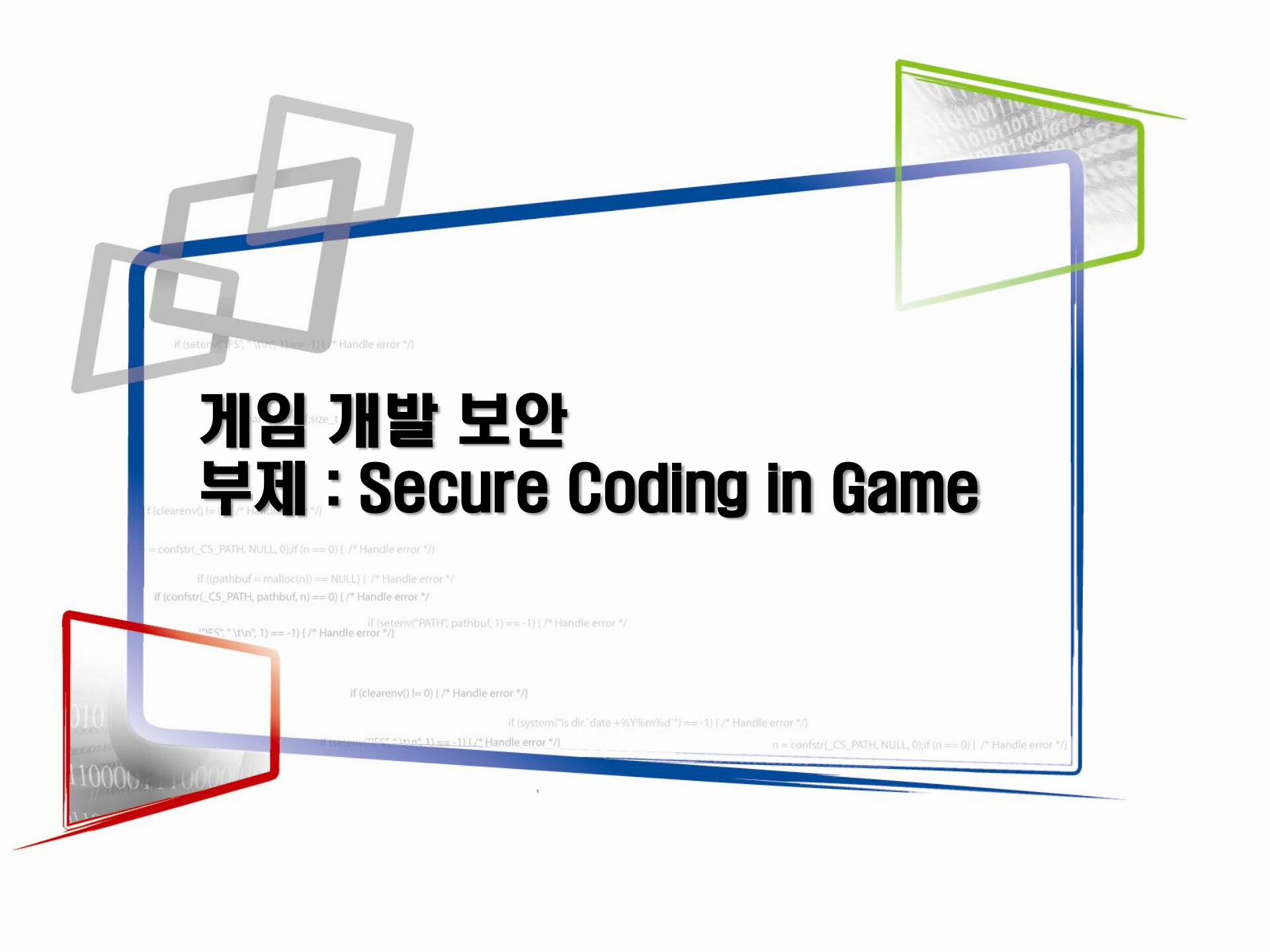
게임 개발 보안
부제 : Secure Coding in Game



강사 소개

이 름 : 이재춘

E 메일 : lhero@hanmail.net

- (현) 한국인터넷진흥원
  - 소스코드 취약성 검증
  - CC 평가

- 시네픽스 게임사업부
  - 아쿠아키즈(PS2 게임) 개발
- 보이시안 닷컴
  - 서버개발

KISA
한국인터넷진흥원

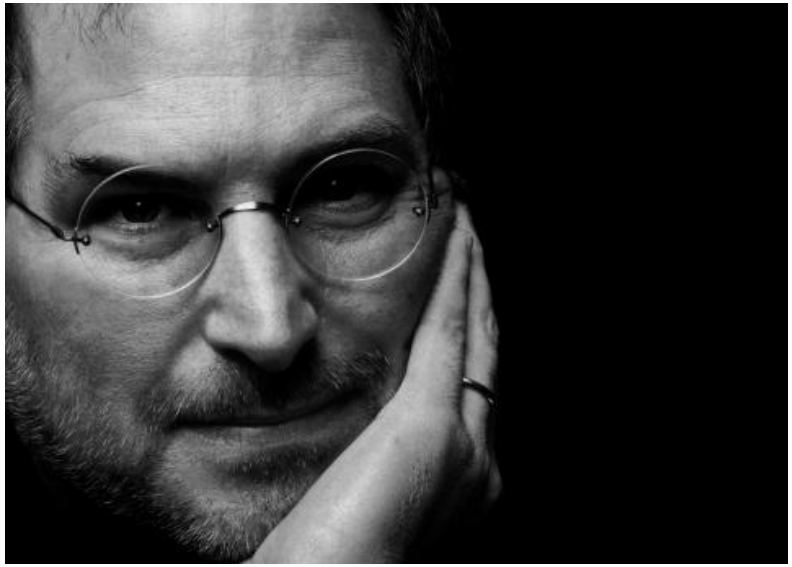
Chapter

1 게임 보안의 파라다임

2 Secure Coding in Game

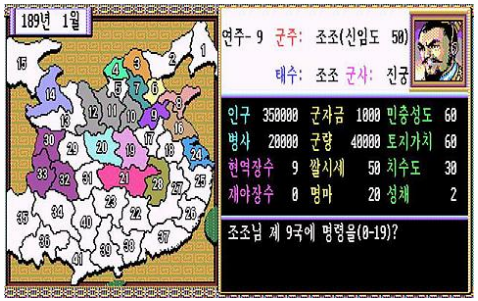
KISA
한국인터넷진흥원

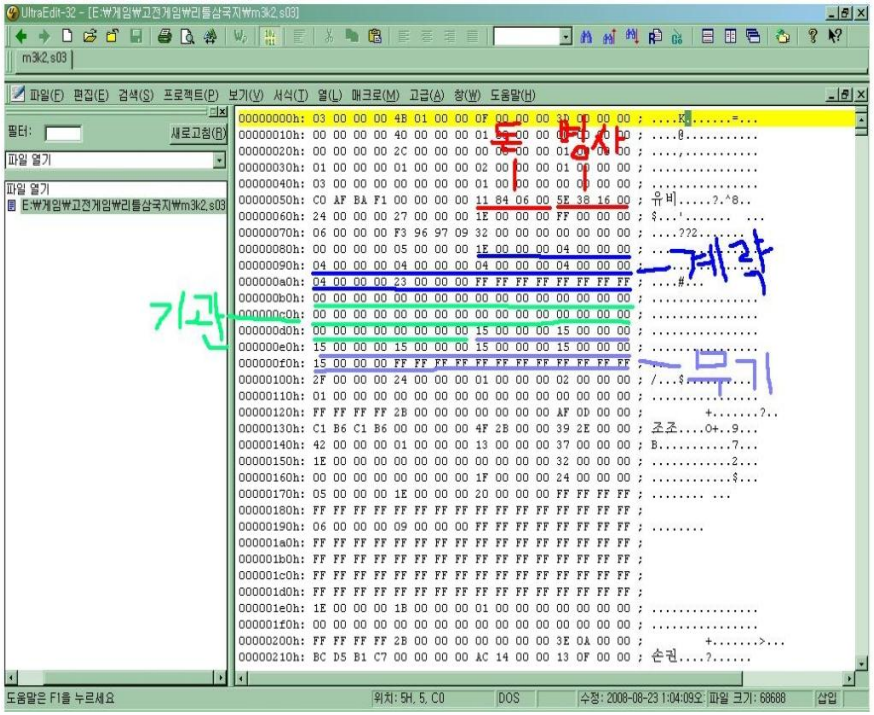
게임 보안의 파라다임



게임해킹의 파라다임(1/3)

- 1세대(PC) 해킹 기법
  - 파일 크랙
  - 메모리 조작





KISA
한국인터넷진흥원

게임해킹의 파라다임(2/3)

- 2세대(온라인 게임) 해킹 기법
  1) 서버
  - 게임소스 탈취 -> 프리서버
  - Dos, 윔, 패킷 리플라이
  - Non Client Bot
  2) 클라이언트
  - 스피드 핵
  - 메모리/파일 변조
    (Dll injetion)
  - 패킷 변조
  - 툴(ex. 조준핵)
  - 계정 해킹

게임해킹의 파라다임(3/3)

- 3세대(모바일 + 온라인 게임 : SNS 게임 등장) 해킹 기법
  - 새로운 이슈들의 등장
  1) 웹 보안
  2) 플랫폼의 Open API
  3) 플랫폼간의 data 통신
  4) middleware 취약점
     (flash 취약점 등)

모바일
SNS 게임

온라인 게임

웹게임

KISA
한국인터넷진흥원

사례1. 해킹 도구 공유 사이트

- 누구나 쉽게 게임 해킹 도구 획득 가능
- 검색어 금지 등은 사후 정책적 관리임
- 해외 도구들에 공격은 법적으로 해결할 수 없음

- 대상 : WOW, L2, 길드워 등
- 취약점, 해킹, 봇, 매크로 등 제공

KISA 한국인터넷진흥원

사례2. 쉽게 제작 가능한 게임 핵

- 프로그래밍 언어 지식이 있으면, 핵 제작이 어렵지 않음
- 각종 해킹 지식이 온라인상에 존재

중고생들 온라인게임 '핵' 만들어 판매

연합뉴스 | 기사입력 2011-09-15 10:12

(서울=연합뉴스) 김계연 기자 = 서울 혜화경찰서는 15일 온라인게임의 기능을 마음대로 바꾸는 이른바 '핵' 프로그램을 제작해 판매한 혐의(정보통신망 이용촉진 및 정보보호 등에 관한 법률 위반 등)로 이모(16)군 등 5명을 불구속 입건했다.

이군 등은 6월 초부터 최근까지 '서든어택' 등 온라인 사격게임 프로그램의 소스를 바꿔 총알을 무한대로 쏘거나 조준점을 자동으로 조정하는 일명 '월핵'(wall hack) 프로그램을 만들어 게임 마니아 30여명에게 3만~5만원씩 받고 판 혐의를 받고 있다.

조사결과 이들은 프로그램에 '인증키'를 걸어놓고 팔자마자 비밀번호를 바꿔 프로그램을 못 쓰게 만드는 수법으로 사기행각을 벌이기도 한 것으로 드러났다.

과거 게임해킹의 대응하는 모습

외부공격자

interface
외부 - 내부

사후 해결

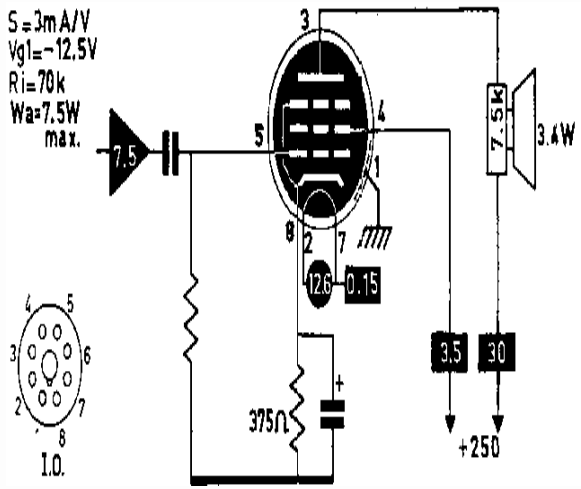
1. 로그 분석

2. 침해 대응

…

KISA
한국인터넷진흥원

필요한 게임 해킹의 대응 자세 – 개발보안



interface

Interface 설계

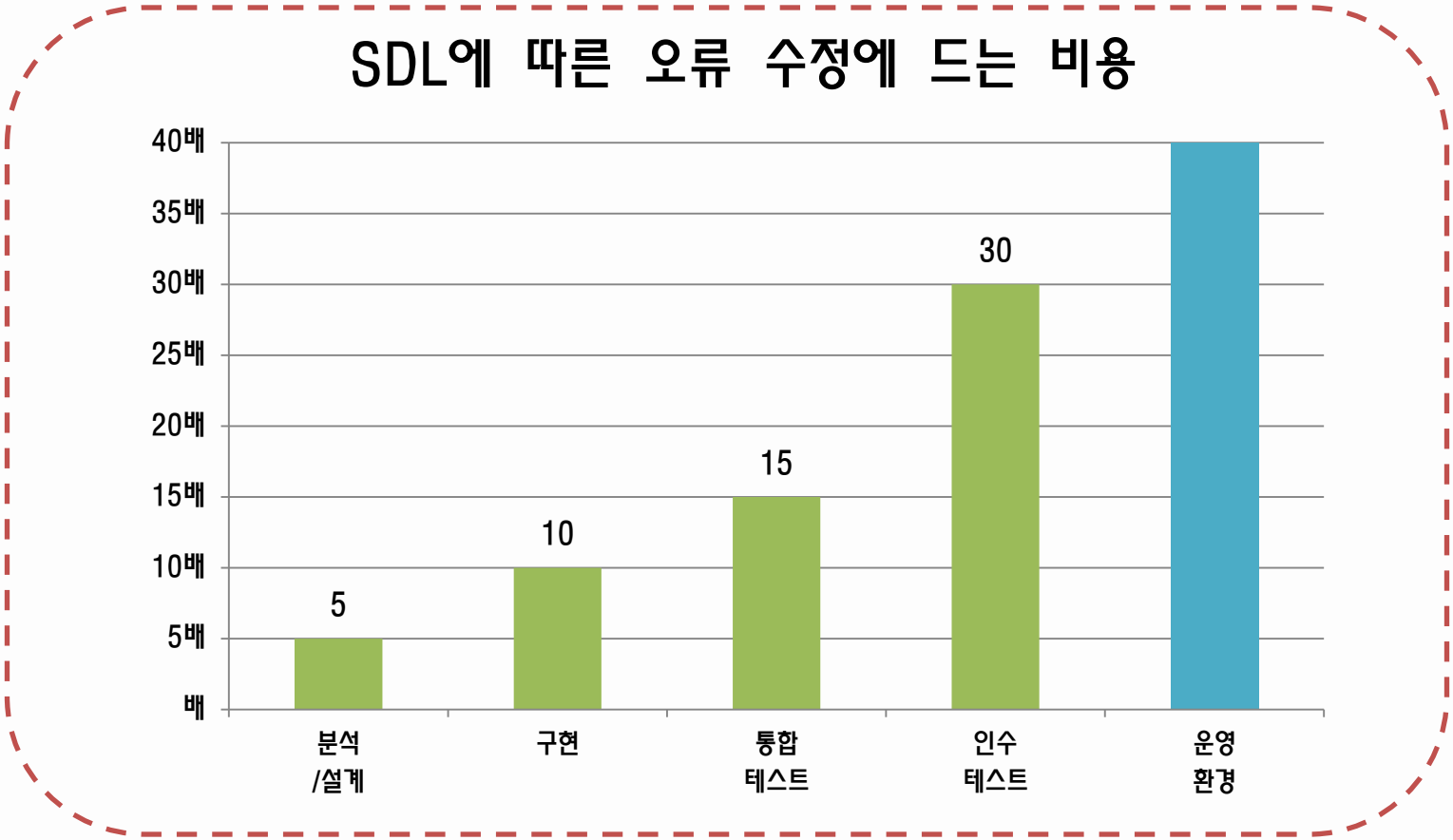
인터페이스 함수 F(X) 에 대하여

1. F(X) 는 취약한 함수를 사용하지 않은가?
2. F(X) 의 X는 입력값 검증이 되어 있는 것인가?
3. 전화선을 통해 주고 받는 중요 데이터 X' 는 암호화 되어 있는가?

…

KISA 한국인터넷진흥원

오류 수정에 드는 비용



KISA
한국인터넷진흥원

보안은 언제 하는것인가?

누가 보안을 생각해야 하는가?

- 보안 관리팀
- 개발자

언제 시작해야 하는가?

- 코드 작성 시작 시
- 빌드 시

결과 확인 후 조치?

- 산출물 피드백
- 결과 통합 관리

Security의 책임

보안 위협을 최소화할 책임은?

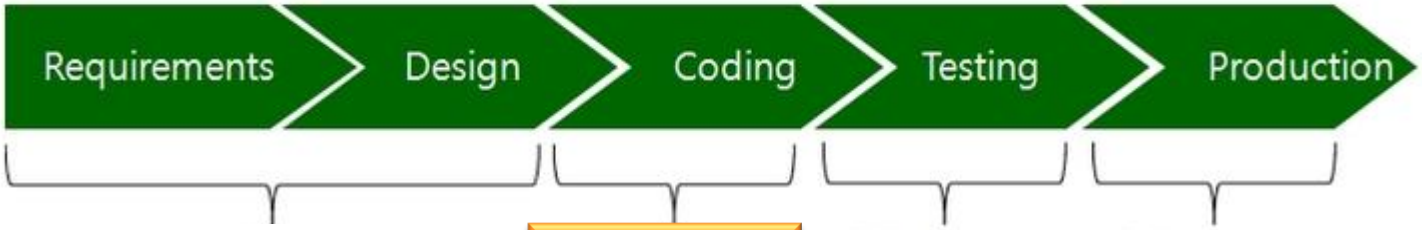
Past

서버 관리자/
운영팀/
보안관리팀

New

프로그래머

개발보안



--------< 분석 > ----------

1. 장르분석 : MMO, FPS
   -> FPS : 에임 핵, …
   -> RPG : 오토, 스피드핵
2. 인터페이스 분석
   -> 웹 기반 통신
   -> C/S 통신
   -> SNS

-------<대응 > ------------

1. 봇 방지 : captcha, 패턴 리포트
2. 스피드 핵 : 시간함수 및 로그자동 분석 리포드
3. 웹 기반 : XSS 방지 등

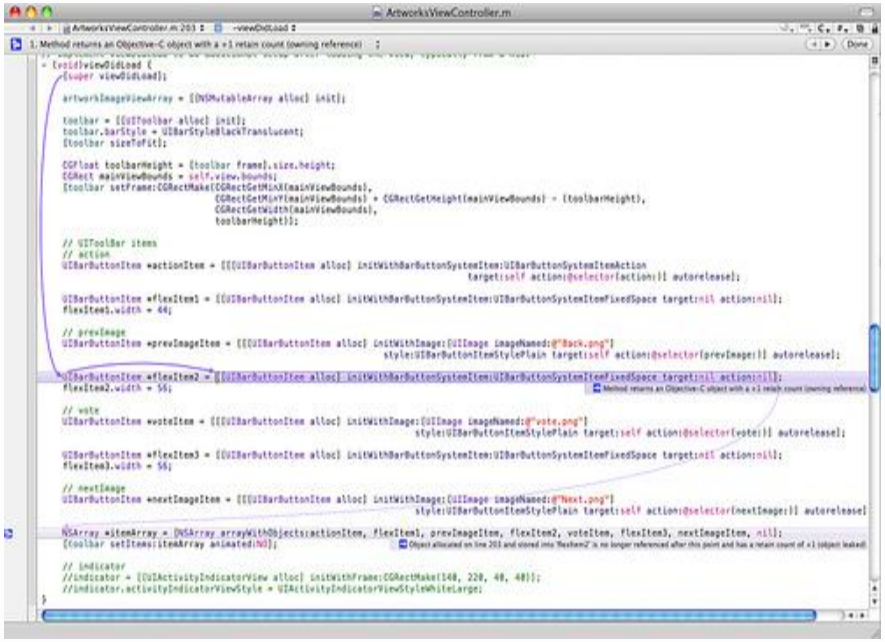
Secure Coding

동적 분석 (실행 영역 보안 취약점 분석)

동적 분석, 침해 모니터링 및 차단

KISA 한국인터넷진흥원

Secure Coding in Game



KISA
한국인터넷진흥원

Seven Pernicious Kingdom

- 취약점을 프로그래밍 개념을 기준

① 입력데이터 검증 및 표현
(Input validation and representation)
② API 오용(API abuse)
③ 보안특성(Security features)
④ 시간 및 상태(Time and state)
⑤ 에러처리(Errors)
⑥ 코드품질(Code quality)
⑦ 캡슐화(Encapsulation)
* 환경(Environment)

http://cwe.mitre.org/documents/sources/SevenPerniciousKingdoms.pdf

KISA
한국인터넷진흥원

1. 입력데이터 검증 및 표현

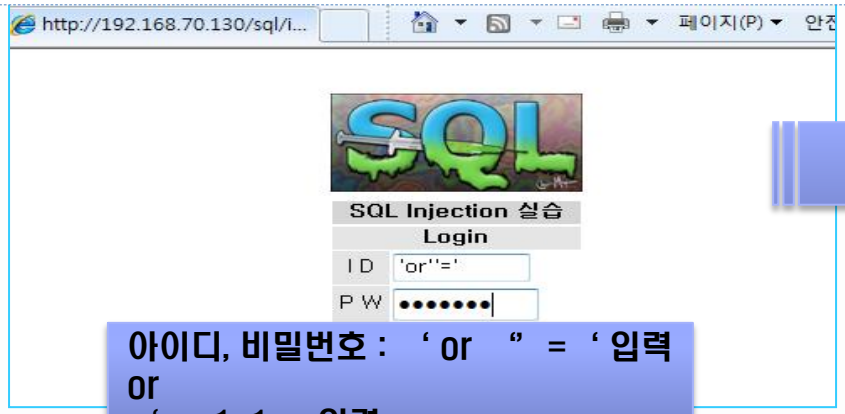
- 개발시 프로그램 입력값에 대한 검증 누락 또는 부적절한 검증이나 사용되는 데이터의 잘못된 형식지정으로 인해 발생할 수 있는 보안취약점.

- XSS, SQL 삽입, 버퍼 오버플로우, OS 명령어 삽입 공격

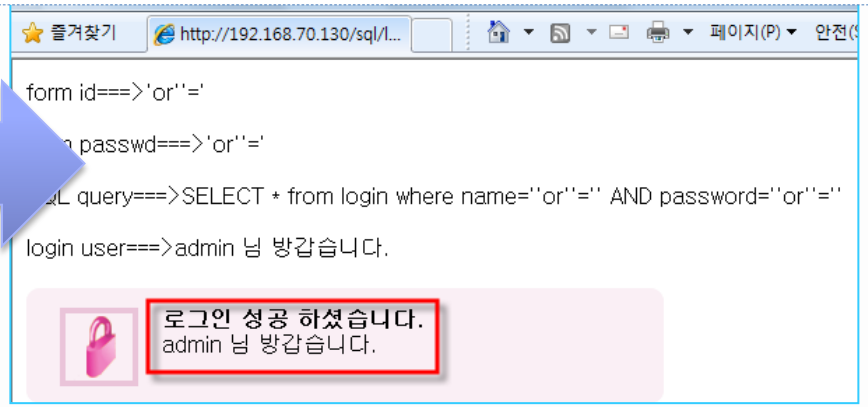
SQL 삽입(1/5)

정의

공격자가 외부 입력을 통해서 SQL 명령어를 수행할 수 있다. 즉 외부 입력한 데이터에 대한 유효성을 점검하지 않아 쿼리 로직이 변경 되어 공격자의 의도대로 타인의 정보 유출 또는 DB의 변경이 발생할 수 있다.



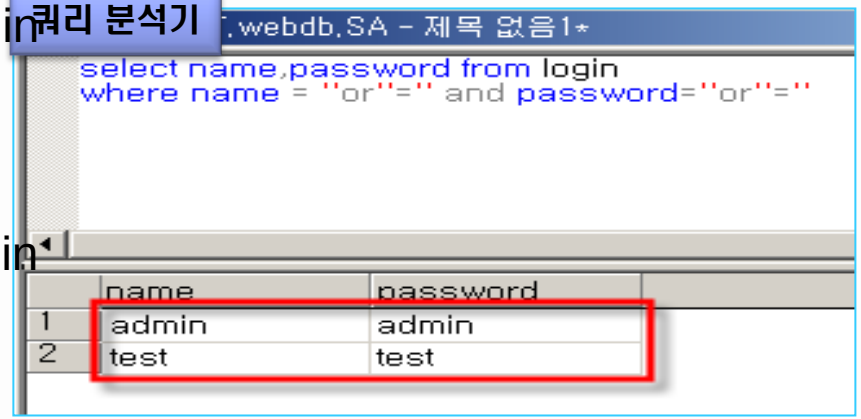
아이디, 비밀번호 : ‘ or ‘’ = ‘ 입력
or
‘ or 1=1-- 입력



strSQL="select name, password from login
Where name=‘"&id&“’ and
password=‘"&password&“’"

strSQL="select name, password from login
Where name=‘’ or ‘’=‘’ and
password=‘’ or ‘’=‘’”

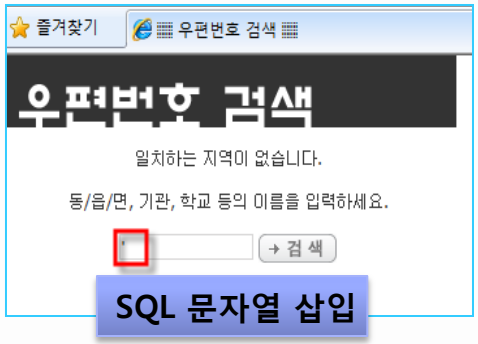
쿼리 분석기

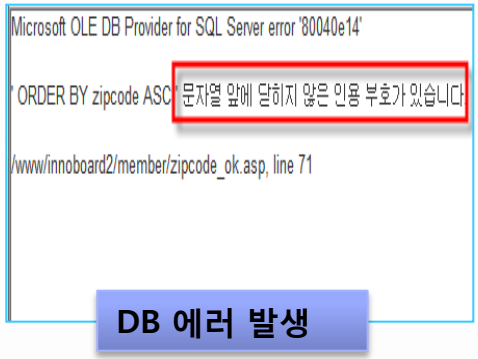


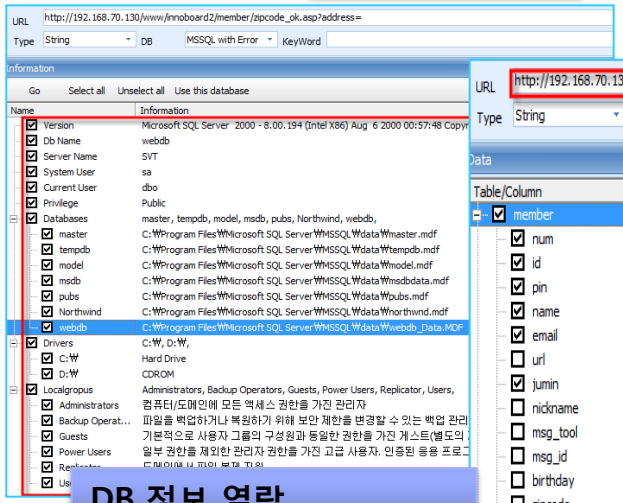
KISA 한국인터넷진흥원

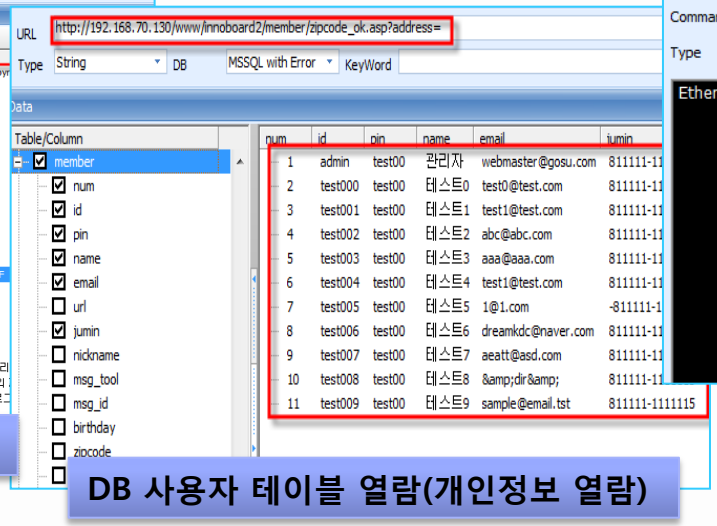
SQL 삽입 (2/5)

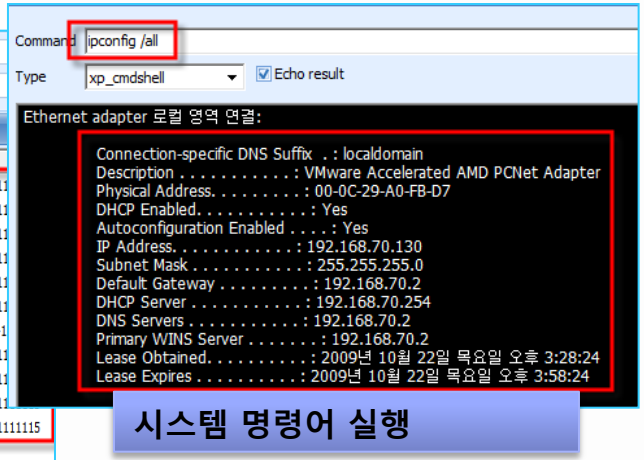
◆ 자동화 공격 도구를 이용한 SQL Injection 공격











KISA 한국인터넷진흥원

SQL 삽입 (3/5)

안전하지 않은 코드

```
1:  ……
2:  PreparedStatement stmt = null;
3:
4:  try {
5:          ……
6:          // 외부 환경에서 테이블명(tablename)과 사용자명(name)을 입력받는다.
7:          String tableName = props.getProperty("jdbc.tableName");
8:          String name = props.getProperty("jdbc.name");
9:          String query = "SELECT * FROM " + tableName + " WHERE Name =" + name;
10:
11:
12:
13:         stmt = con.prepareStatement(query);
14:         rs = stmt.executeQuery();
15:         ResultSetMetaData rsmd = rs.getMetaData();
16:         ……
17:         while (rs.next()) {    ……    }
18:         dos.writeBytes(printStr);
19: } catch (SQLException sqle) {    ……    }
20: finally {    ……    }
21: ……
```

SQL 삽입 (4/5)

```
"SELECT * FROM " + tableName + " WHERE Name = '" + name + "'"
```

tableName: userTable name: gildong

"SELECT * FROM userTable WHERE Name = 'gildong'"

tableName: userTable name: name' OR 'a'='a

"SELECT * FROM userTable WHERE Name = 'name' OR 'a'='a'"

tableName: userTable name: name'; DELETE FROM userTable; --

"SELECT * FROM userTable WHERE Name = 'name'; DELETE FROM userTable; --'"

SQL 삽입 (5/5)

안전한 코드

```
1:  ……
2:  PreparedStatement stmt = null;
3:
4:  try {
5:          ……
6:          String tableName = props.getProperty("jdbc.tableName");
7:          String name = props.getProperty("jdbc.name");
8:
9:          // 동적 질의문으로 생성되지 않도록 PreparedStatement를 사용한다.
10:         String query = "SELECT   * FROM  ? WHERE Name = ? ";
11:         stmt = con.prepareStatement(query);
12:         // 사용자가 입력한 데이터를 setXXX()함수로 셋팅한다.
13:         stmt.setString(1, tableName);
14:         stmt.setString(2, name);
15:
16:         rs = stmt.executeQuery();
17:         ResultSetMetaData rsmd =   rs.getMetaData();
18:         int columnCount =   rsmd.getColumnCount();
19:         String printStr =   "";
20:         while (rs.next()) {   ……   }
21:         dos.writeBytes(printStr);
22: } catch (SQLException sqle) {   ……   }
23: finally {   ……   }
24: ……
```

정수 오버플로우 (1/2)

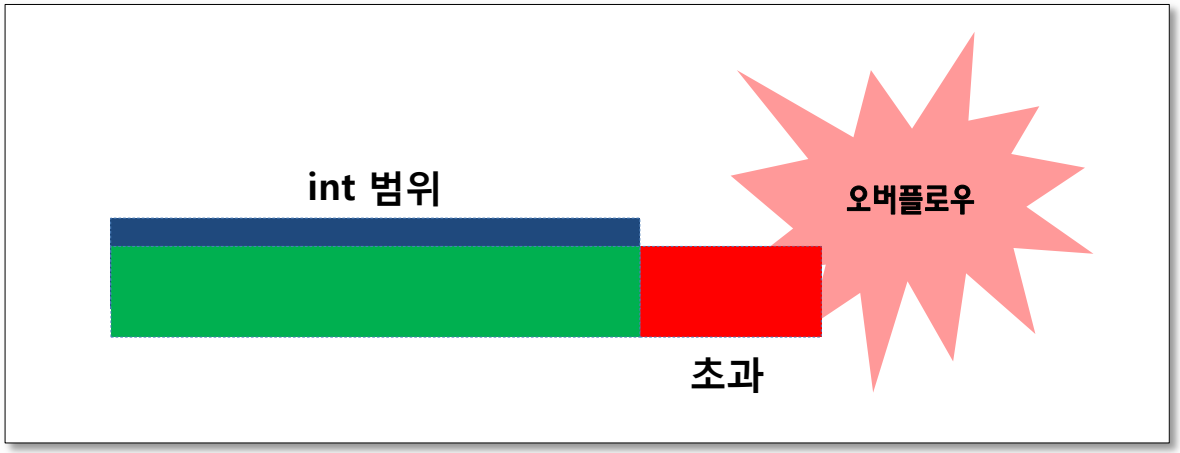
정의

정수형 변수의 오버플로우는 정수값이 증가하면서, Java에서 허용된 가장 큰 값보다 더 커져서 실제 저장되는 값은 의도하지 않게 아주 작은 수이거나 음수가 될 수 있다. 이러한 상황을 검사하지 않고 그 값을 순환문의 조건이나 메모리 할당, 메모리 복사 등에 쓰거나, 그 값에 근거해서 보안 관련 결정을 하면 취약점을 야기할 수 있다.

```
int a = 2147483647;
int b = 2147483647;

System.out.println(a+b);
```

→ **-2**



정수 오버플로우 (2/2)

안전하지 않은 코드

```
1:  ……
2:  public static void main(String[] args) {
3:      int size = new Integer(args[0]).intValue();
4:              size +=  new Integer(args[1]).intValue();
5:              MyClass[] data = new MyClass[size];
6:  ……
7:
```

안전한 코드

```
1:  ……
2:  public static void main(String[] args) {
3:      int size = new Integer(args[0]).intValue();
4:      size +=  new Integer(args[1]).intValue();
5:              // 배열의 크기 값이 음수값이 아닌지 검사한다.
6:              if (size < 0) return ;
7:              MyClass[ ] data = new MyClass[size];
8:  ……
```

안전한 코딩기법

동적 메모리 할당을 위해서 사용되는 변수가 이전 처리 과정에서 오버플로우에 의해서 음수값으로 변환될 수 있으므로 미리 변수의 값 범위를 검사한다.

2. API 오용

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

  1) 보안취약점의 이유로 금지된 함수들(MS)

     - memcpy(), copymemory(), rtlcopymemory()

  2) 사용환경의 변화로 사용이 금지된 API

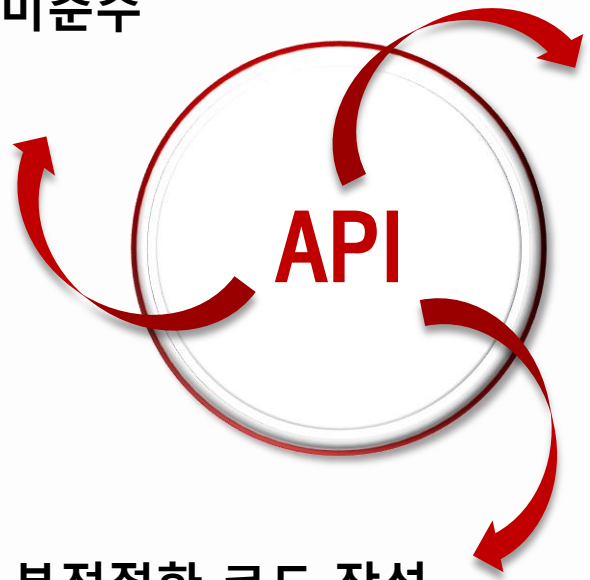
     - J2EE: System.exit(), J2EE:Main()

  3) 취약함으로 인한 함수 변경

     - Scanf() => Scanf_s()

API 오용

API 작성 규칙 미준수

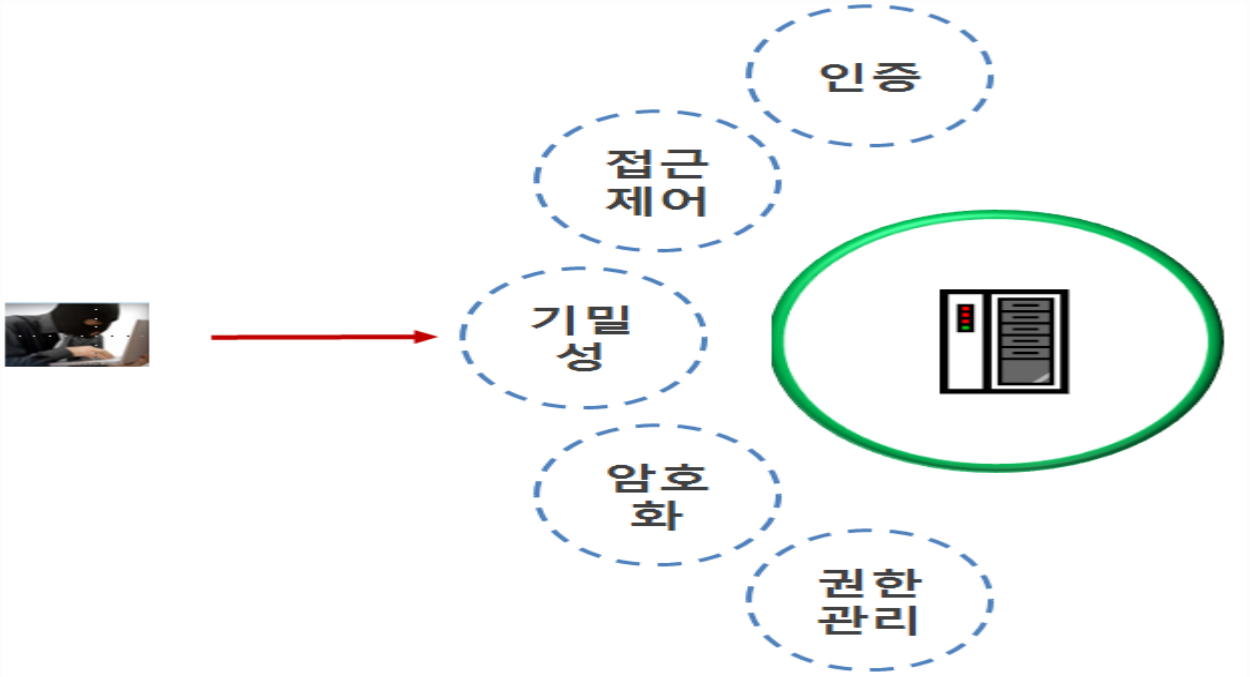


API 이해 부족

부적절한 코드 작성

3. 보안특성

- 보안특성(인증, 접근제어, 기밀성, 암호화, 권한 관리 등)을 부주의하게 구현시 발생할 수 있는 보안취약점.
- 대표적으로 부적절한 인가, 하드코드된 패스워드, 취약한 암호화 알고리즘 사용 등



하드코드된 패스워드(1/3)

정의

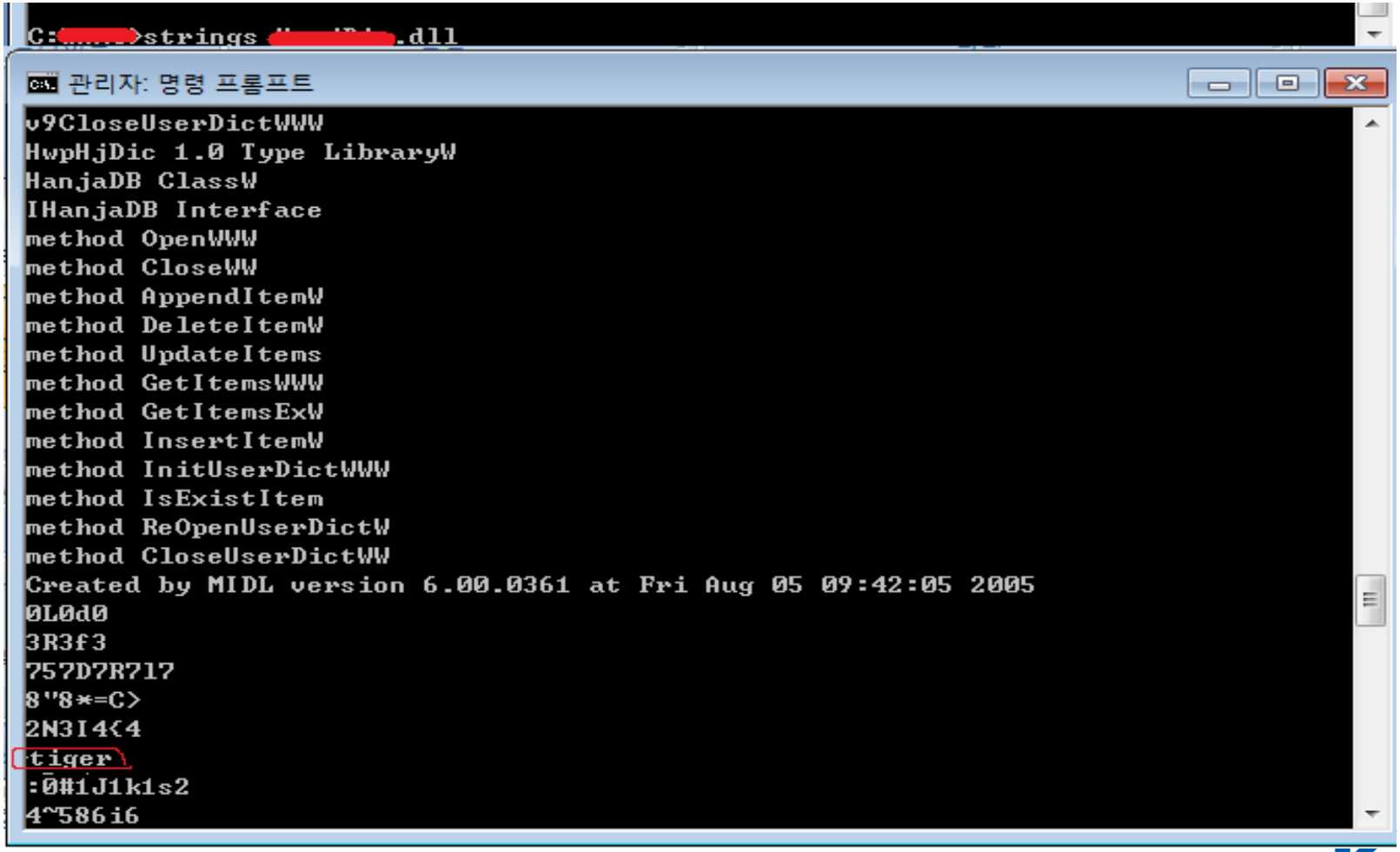
소프트웨어가 코드 내부에 하드코드된 패스워드를 포함하고, 이를 이용하여 내부 인증에 사용하거나 외부 컴포넌트와 통신을 하는 것은 위험하다.
클라이언트를 hex 에디터 등을 통하여 해당 아이디 패스워드 탈취가 가능하다.

안전하지 않은 코드

```
1:   public class U259 {
2:       private Connection conn;
3:
4:       public Connection DBConnect(String url, String id) {
5:           try {
6:
7:               conn =    DriverManager.getConnection(url, id, "tiger");
8:           } catch (SQLException e) {
9:               System.err.println("...");
10:          }
11:          return conn;
12:      }
13:           ……
14:  }
```

하드코드된 패스워드 (2/3)

C:>Strings 타겟 파일
⇒아스키 값, unicode 값을 실행파일에서 읽어 오는 도구(MS 테크넷에서 다운 가능)



KISA
한국인터넷진흥원

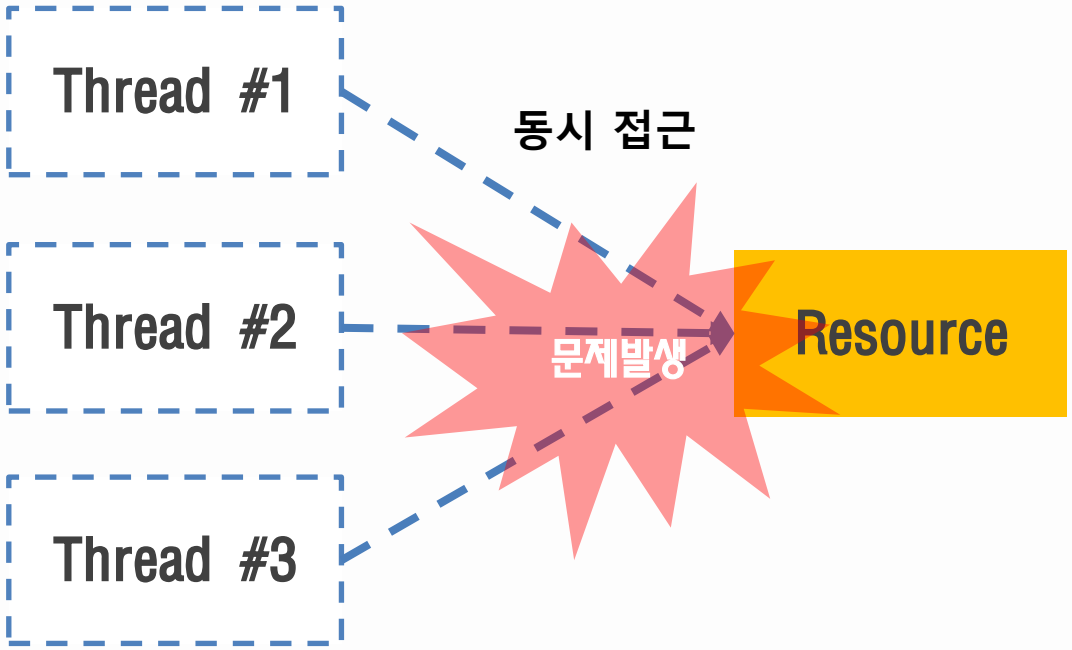
하드코드된 패스워드 (3/3)

안전한 코드

```
1:  public class S259 {
2:          public Connection connect(Properties props) throws NoSuchAlgorithmException,
                        NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException,
                                                                        BadPaddingException {
3:          try {
4:                  String url = props.getProperty("url");
5:                  String id = props.getProperty("id");
6:                  String pwd = props.getProperty("passwd");
7:
8:          //외부 설정 파일에서 패스워드를 가져오며, 패스워드가 값이 있는지 체크하고 있음
9:                  if (url != null && !"".equals(url) && id != null &&   !"".equals(id)
10:                                                  && pwd != null &&   !"".equals(pwd)) {
11:                 KeyGenerator kgen = KeyGenerator.getInstance("Blowfish");
12:                 SecretKey skey = kgen.generateKey();
13:                 byte[] raw = skey.getEncoded();
14:                 SecretKeySpec skeySpec = new SecretKeySpec(raw, "Blowfish");
15:
16:                 Cipher cipher = Cipher.getInstance("Blowfish");
17:                 cipher.init(Cipher.DECRYPT_MODE, skeySpec);
18:                 byte[] decrypted_pwd = cipher.doFinal(pwd.getBytes());
19:                 pwd = new String(decrypted_pwd);
20:                 conn = DriverManager.getConnection(url, id, pwd);
21:         }
22:     } catch (SQLException e) {
23:  ……
```

4. 시간 및 상태

- 동시 또는 거의 동시 수행을 지원하는 병렬 시스템, 프로세스 또는 스레드 환경에서 시간 및 상태를 부적절하게 관리하여 발생할 수 있는 보안취약점.
- 대표적으로 데드락(dead lock), 자원에 대한 경쟁조건, 세션 고착 등



KISA
한국인터넷진흥원

코드 정확성: 동기화된 메소드를 비동기화된 메소드로 재정의

안전하지 않은 코드

```
1:   class FileMgmtThread extends Thread {
2:       private String manageType = "";
3:       public FileMgmtThread (String type) {
4:          manageType = type;
5:       }
6:      public void run() {
7:        try {
8:           if ( manageType.equals("READ") ) {
9:              File f = new File("Test_367.txt");
10:               if (f.exists()) {  // 만약 파일이 존재하면 파일내용을 읽음
11:                  BufferedReader br = new BufferedReader(new FileReader(f));
12:                  br.close();
13:               }
14:            } else if ( manageType.equals("DELETE") ) {
15:               File f = new File("Test_367.txt");
16:               if (f.exists()) {  // 만약 파일이 존재하면 파일을 삭제함
17:                  f.delete();
18:               } else {
19:                  ;
20:               }
21:            }
22:        } catch (IOException e) {
23:        }
24:      }
25:   }
```

코드 정확성: 동기화된 메소드를 비동기화된 메소드로 재정의

안전하지 않은 코드

```
26:   public class CWE367 {
27:     public static void main(String[] args) {
28:           // 파일의 읽기와 파일을 삭제하는 것을 동시에 수행한다.
29:           FileMgmtThread fileAccessThread = new FileMgmtThread("READ");
30:           FileMgmtThread fileDeleteThread = new FileMgmtThread("DELETE");

31:           fileAccessThread.start();
32:           fileDeleteThread.start();
33:     }
34:   }
```

코드 정확성: 동기화된 메소드를 비동기화된 메소드로 재정의

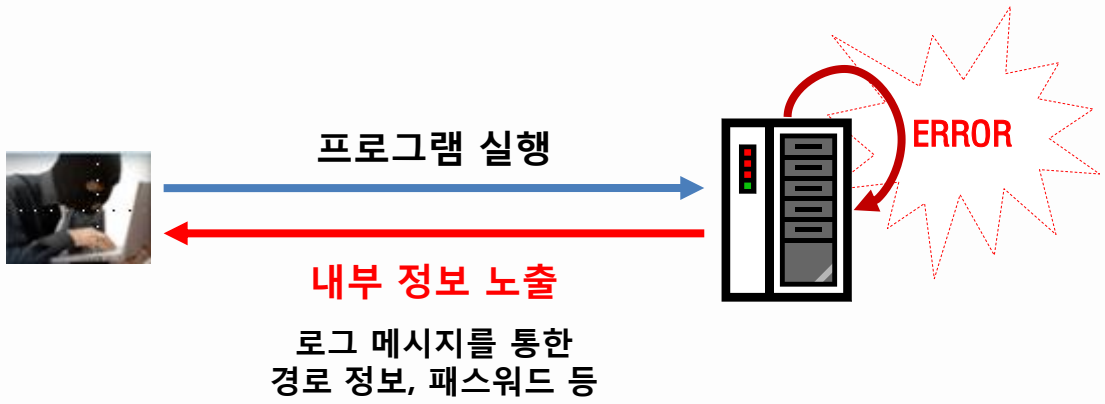
안전한 코드

```
1:  class FileMgmtThread extends Thread {
2:      private static final String SYNC = "SYNC";
3:      private String manageType = "";
4:      public FileMgmtThread (String type) {
5:         manageType = type;
6:      }
7:     public void run() {
8:       synchronized(SYNC) {
9:         try {
10:            if ( manageType.equals("READ") ) {
11:               File f = new File("Test_367.txt");
12:               if (f.exists()) {  // 만약 파일이 존재하면 파일내용을 읽음
13:                  BufferedReader br = new BufferedReader(new FileReader(f));
14:                  br.close();
15:               }
16:            } else if ( manageType.equals("DELETE") ) {
17:               File f = new File("Test_367.txt");
18:               if (f.exists()) {  // 만약 파일이 존재하면 파일을 삭제함
19:                  f.delete();
20:               } else { }
21:            }
22:         } catch (IOException e) {
23:       }
24:     }
25:   }
26: }
```

synchronized 를 사용하면 지정된 객체에 lock이 걸려서 해당 블록을 수행하는 동안 다른 Thread가 접근할 수 없다.

5.에러처리

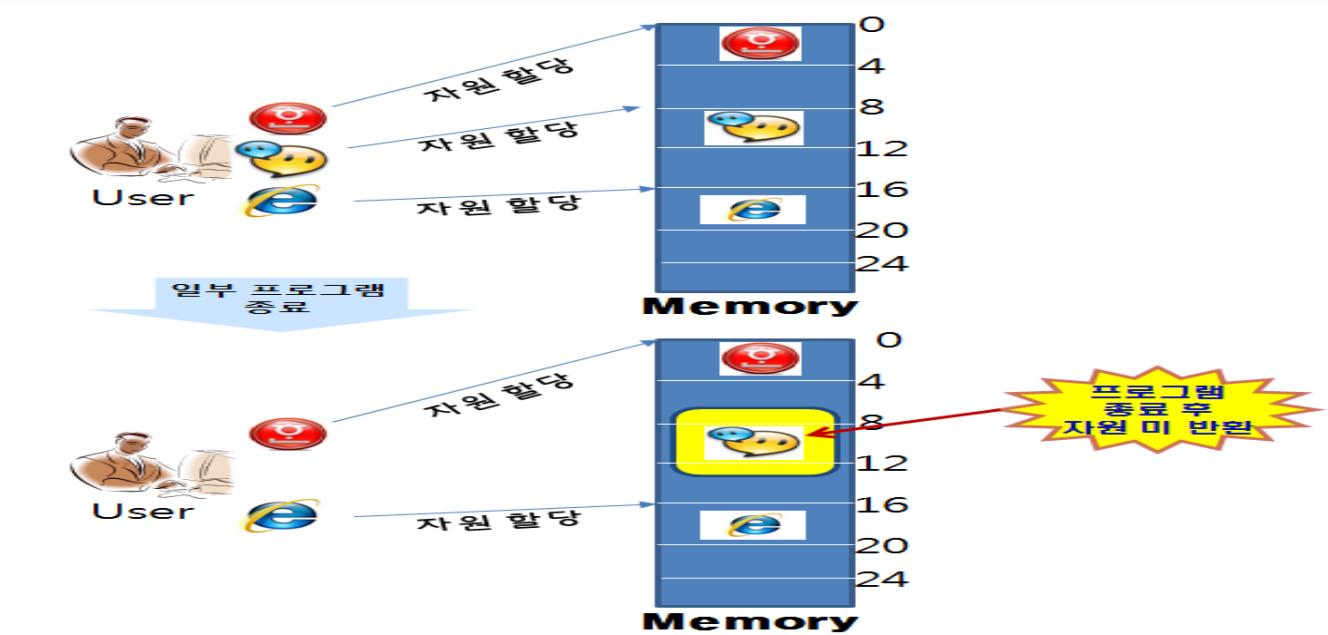
- 에러를 불충하게 처리하거나 전혀 처리를 하지 않거나 에러 정보에 과도하게 많은 정보가 포함될 때 발생할 수 있는 보안 취약점.
- 대표적으로 에러처리 루틴의 누락, 에러처리 시 필요이상의 정보 노출 등



KISA
한국인터넷진흥원

6.코드품질

- 복잡한 소스코드로 인해 관리성, 유지보수성, 가독성이 저하되어 SW 개발. 유지보수시 타입변환 오류, 자원(메모리 등)의 부적절한 반환 등과 같이 개발자가 범할 수 있는 코딩오류로 인해 유발되는 보안취약점.
- 대표적으로 자원의 부적절한 반환, 무한할당 등



KISA
한국인터넷진흥원

무한 자원 할당(1/2)

정의

프로그램이 자원을 사용 후 해제하지 않거나, 한 사용자당 서비스할 수 있는 자원의 양을 제한하지 않고, 서비스 요청마다 요구하는 자원을 할당한다.

안전하지 않은 코드

```
1: Connection conn = null;
2: PreparedStatement pstmt = null;
3: try {
4:              conn=getConnection();
5:    ...
6:    pstmt = conn.prepareStatement("SELECT * FROM employees
7:        where name=?");
8:    ...
9:    conn.close();
10:    pstmt.close();
11: }catch (SQLException ex) {...}
```

무한 자원 할당 (2/2)

안전한 코드

```
1: Connection conn = null;
2: PreparedStatement pstmt = null;
3: try {
4:              conn=getConnection();
5:     ...
6:     pstmt = conn.prepareStatement("SELECT * FROM employees
7:         where name=?");
8:     ...
9: }  catch (SQLException ex) {...}
10: // 자원을 사용하고 해제 시 항상 finally문에서 한다.
11: finally {
12:   if ( conn!= null ) try { conn.close(); } catch (SQLException e){...}
13:    if ( pstmt!= null ) try { pstmt.close(); } catch (SQLException e){...}
14:}
```

안전한 코딩기법

- 프로그램에서 자원을 오픈하여 사용하고 난 후, 반드시 자원을 해제한다.
- 사용자가 사용할 수 있는 자원의 사이즈를 제한한다.

※ 사용자가 접근할 수 있는 자원의 양을 제한한다. 특히 제한된 자원을 효율적으로 사용하기 위해서 Pool(Thread Pool, Connection Pool 등)을 사용한다.

7.캡슐화

- 중요한 데이터 또는 기능성을 불충분하게 캡슐화하였을 때 인가되지 않은 사용자 또는 시스템에게 데이터 누출이 가능해지는 보안취약점.
- 대표적으로 제거되지 않고 남은 디버거 코드, 시스템 데이터 정보누출 등

공용 메서드로부터 리턴된 private 배열-유형 필드

정의

private로 선언된 배열을 public으로 선언된 메소드를 통해 반환(return)하면, 그 배열의 레퍼런스가 외부에 공개되어 외부에서 배열의 수정할 수 있다.

안전하지 않은 코드

```
1:   ……
2:   private String[] colors = {"b", "b", "b"};

3:   public String[] getColors() {    return colors;    }
4:   ……
```

```
String[] colors2 = getColors();
colors2[0] = "a";

이러한 코드가 있을 때, colors[0]은?
```

이 메소드를 호출한 쪽에서
배열의 내용을 수정할 경우
원본 배열인 colors의 내용도 변경됨

공용 메서드로부터 리턴된 private 배열-유형 필드

안전한 코드

```
1:  ......
2:  private String[] colors = {"b", "b", "b"};
3:
4:  public String[] getColors() {
5:      String[] ret = null;
6:      if ( this.colors != null ) {
7:                  ret = new String[colors.length];
8:                  for (int i = 0; i < colors.length; i++) {    ret[i] = this.colors[i];    }
9:      }
10:     return ret;
11: }
12: ......
```

내용이 동일한 복제본

TIP & TECH

일반적으로 for문은 성능이 느리므로 System.arraycopy() 메소드를 많이 사용함

안전한 코딩기법

private로 선언된 배열을 public으로 선언된 메소드를 통해 반환하지 않도록 해야 한다. 필요한 경우 배열의 복제본을 반환하거나, 수정을 제어하는 public 메소드를 별도로 선언하여 사용한다

질의 및 응답

감사합니다

KISA
한국인터넷진흥원