令和元(2019)年度特別研究費 実績報告書

令和2年3月20日

公立千歳科学技術大学
学長　川瀬　正明　様

公立千歳科学技術大学特別研究等助成要綱第 4 条に基づき，下記のとおり報告いたします.

採択区分	■若手研究者支援研究費　　□基盤的研究費			
報告者	所属	理工学部	職名	助手
	氏名	砂原 悟	ふりがな	すなはら　さとる
研究課題名	静的プログラム解析を用いたインジェクション系の脆弱性検出に対する有効性の評価			
本研究費による発表論文，著書など	無し			

研究成果報告

(1) 研究の背景と目的

日本国内においてインターネットの利用者が 8 割を超え[1]，ネットショッピングやインターネットバンキング，音楽配信などのサービスが非常に身近となった一方で，Web アプリケーションの脆弱性(悪用されかねない不具合)は年間 10000 万件を超える報告があり，実際に悪用された事例が複数報告されている[2][3][4]．脆弱性の中でもインジェクションは過去数年間，最も重大なセキュリティリスクと報告 [5]されている．また，IPA(情報処理推進機構)の 2018 年度四半期の報告でも Web サイトにおけるインジェクションの届出は昨年比 2.5 倍となっており，有効的な対策が進んでいない分野といえる．

インジェクションの脆弱性を Web アプリケーションに紛れ込ませないようにするにはプログラム上の外部入力点からデータベース等のセキュリティ上考慮されるべき出力点(Sink; シンク)までのデータフローを解析し，各フローでの出力点応じたエスケープ処理(Sanitizer; サニタイザ)が適用されている必要がある．また，適切なエスケープ処理が行われているかを見落とさないためにはリリース前のコードレビューが有効であるが，コードレビューを任せられる要員を確保することが困難なケースや十分な時間を確保できない場合もあり，機械的に精度良くシンクを発見できること，また，レビュー時間を短縮できることが開発におけるセキュリティ対策として重要である．

本研究では，シンクを機械的に発見するためにどのような課題があるのかを明らかにするために，ソースコード解析ツールである SAST(Static Application Security Testing)を用いて既知のインジェクションをどの程度発見できるかの精度調査と，発見できないケースの分析を行った．

(2) 本研究で用いた SAST ソフトウェアの特徴と調査対象

解析には複数のプログラミング言語(Java, Ruby, Python, PHP, Javascript)に対応し，SQL インジェクション，OS コマンドインジェクション，XSS，CSRF などのインジェクションを検知した実績がある SAST ソフトウェアを用いた．

調査対象は，日本で使用されるソフトウェアなどの脆弱性関連情報が報告されている JVN(Japan Vulnerability Notes)[6]において，2005 年 9 月から 2020 年 1 月までにインジェクション(CWE-74)と分類されたソフトウェアから優先度「緊急」である 90 件のうち，2020 年 1 月現在でも無償でソースコードが手に入り，かつ，SAST ソフトウェアで解析可能であった 50 件である．

(3) 分析の結果とプログラムの特徴について

分析の結果，報告されている既知の脆弱性を検知できた件数は 50 件中 1 件(誤検知は 0 件)であった．検知できたのは JVNDB-2019-013013(表 1)の安全ではないデシリアライゼーションで，認証されていないユーザが cookie に対して悪意のある PHP オブジェクトを埋め込むことで任意のコードが実行可能である．

```
$items = !empty($_COOKIE['comparison']) ? unserialize($_COOKIE['comparison']) : array();
```

表 1:JVNDB-2019-013013 安全ではないデシリアライゼーション

JVNDB-2018-013923(表 2)も安全ではないデシリアライゼーションの脆弱性となっているが，こちらは検知されていない．JVNDB-2019-013013 についてはユーザから受け取った値を直接デシリアライズしてしまうため，疑いの余地なくインジェクションが成立してしまうプログラムであるが，JVNDB-2018-013923 については変数$p が事前にエスケープされている可能性があり，誤った判定を避けるために検知されなかったと考えられる．

```
$p = parent::getPref($prefName);
    if (strpos($p, '$phpserial$') === 0) {
        $p = substr($p, strlen('$phpserial$'));
        return unserialize($p);
```

表 2: JVNDB-2018-013923 安全ではないデシリアライゼーション

同様に判定が難しいケースは JVNDB-2013-007002(表 3), JVNDB-2016-009337(表 4), JVNDB-2016-001459(表 5)など，多数存在した.

```
uri_obj = URI.parse(url)
if uri_obj.kind_of? URI::HTTP or uri_obj.kind_of? URI::FTP
    manifest = `curl --max-time #{max_dl_time} --connect-timeout 2 --location --max-redirs
#{max_redirs} --max-filesize #{max_file_size} -k #{url}`
```

表 3: JVNDB-2013-007002 shell のエスケープ漏れによる OS コマンドインジェクション

```
$tag = new expTag($this->params['tag']);
```

表 4: JVNDB-2016-009337 ユーザから受け取った値をエスケープしないことによるリスク

```
String agent = System.getProperty("http.agent");
return agent != null ? agent : ("Java" + System.getProperty("java.version"));
```

表 5: JVNDB-2016-001459 細工された HTTP ヘッダから cookie の値を設定されるリスク

JVNDB-2019-010710(表 6)の事例は，それまでに知られていなかった攻撃方法によって悪用が可能と判明したシンクである．コードレビューにおいて，インジェクションが成立することが明らかとなっていたが，無害であると判断され，シンクに対するエスケープが行われていなかった．この事例のように比較的新しい既知のインジェクションの中にはどのような攻撃手法が成立するか判明していないシンクが存在する．攻撃手法が明らかではない場合，適切なエスケープが行われているか機械的な判定をすることが難しい.

```
<input id="client_id" type="text" required={ flow === PASSWORD }
   value={ this.state.clientId }    data-name="clientId onChange={ this.onInputChange } />
                                          中略
<input id="client_secret" value={ this.state.clientSecret } initialValue={ this.state.clientSecret }
    type="text" data-name="clientSecret" onChange={ this.onInputChange }/>
```

表 6: JVNDB-2019-010710 CSS インジェクションにより OAuth の資格情報を DOM の値を通じて奪取

JVNDB-2017-015708(表 7)の事例では Apache Synapse アプリケーション自体のプログラムだけではなく，アプリケーションが参照している Apache Commons Collections ライブラリの脆弱性が悪用可能であった．この場合は利用しているライブラリが対策されているバージョン以上であるかを機械的に比較することで比較的容易に発見可能であるが，本研究ではアプリケーション自体のプログラムのみを解析の対象としたため，検知することができなかった.

```
<commons.collections.version>3.2.1</commons.collections.version>
```

表 7: JVNDB-2017-015708 脆弱性を有するライブラリを参照する設定

JVNDB-2019-010710 ではプログラム上で CSV を取り込む際に悪意のある数式[7]を取り除く必要があり，扱うデータや扱われるアプリケーションに依存する差異が機械的な検知を難しくしている．

```
$writer = $type === 'xlsx' ? new XlsWriter('php://output') : new CsvWriter('php://output');
$contentType = $type === 'xlsx' ? 'application/vnd.ms-excel' : 'text/csv';
$filename = strtolower($filename.'_'.((new ¥DateTime())->format($dateFormat)).'.'.$type);
if ($writer instanceof CsvWriter) {
    $securedData = [];
    foreach ($sourceIterator as $row) {
        $securedRow = [];
        foreach ($row as $cell) {
            if (in_array($cell[0], ['+', '-', '=', '@'])) {
                $securedCell = ' '.$cell;
            } else {
                $securedCell = $cell;
            }
```

表 7: JVNDB-2019-010710 記号「+,-,=,@」を安全に扱う修正が行われた CSV の取り込み処理

(4) まとめと今後の取り組みについて

本研究ではシンクを機械的に発見するための課題を明らかにするために，JVNDB で報告されている緊急度の高いインジェクションの脆弱性を有するアプリケーション対して，SAST ソフトウェアによる解析を行った．発見できたのは 50 件中 1 件であり，誤検知は 0 件であった．検知できなかった要因はインジェクションが成立するかどうかの不確定性やエスケープ処理が環境に依存する，アプリケーションが利用しているライブラリが問題となるなど，複数存在した．

本研究によって脆弱性の要因が一様ではないということが機械的なシンクの発見を妨げている課題であることが明らかとなった．また，SAST ソフトウェアはプログラムを実行せずに構文解析のみでシンクを分析するため，誤った検知を行わないようにフィルタリングを行った場合，検知率が著しく低下する可能性があることについても課題であると思われる．これらの課題を解決するためには，検知率と誤検知率のバランスをとりながら，多様な要因であるインジェクションのシンクを検知する手法を検討することが必要である．具体的な手法としては，ソフトウェアを実行してシンクを発見する DAST ソフトウェア(ファジング)と DevSecOps[8]による運用を考慮した要求分析手法を組み合わせるなどが考えられるが，その際は SAST ソフトウェアのみの場合よりもコストが増加するため，コストに見合うかどうかの評価も行いたい．

[1] 総務省 インターネット利用率 https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/h28/html/nc252110.html
[2] 脆弱性を悪用された事例 1 https://piyolog.hatenadiary.jp/entry/2020/01/20/172436
[3] 脆弱性を悪用された事例 2 https://cybersecurity-jp.com/news/21453
[4] 脆弱性を悪用された事例 3 http://www.security-next.com/109528
[5] OWASP Top Ten Project https://owasp.org/www-project-top-ten/
[6] 脆弱性対策情報データベース https://jvndb.jvn.jp/index.html
[7] CSV インジェクション https://owasp.org/www-community/attacks/CSV_Injection
[8] 小西 慶浩，大久保 隆夫：運用を考慮したセキュリティ要求分析手法の提案，2020-CSEC-88(12)，pp1-6(2020)