# 脆弱性ゼロをめざしたセキュリティテストツール実現に向けた課題考察

山本和徳[1]　石井一彦[1]　國頭吾郎[1]

**概要：** 本稿では，5G ネットワークを脆弱性がない状態で導入することを目的とし，ソフトウェアの脆弱性を自動で検知する技術を確立するための初期段階の検討として，既存のセキュリティテストツールの課題を考察した結果を報告する．

# On Challenges of a Security Test Tool Towards Software Zero Vulnerabilities

KAZUNORI YAMAMOTO[1]　KAZUHIKO ISHII[1]　GORO KUNITO[1]

## 1. はじめに

セキュリティテストは，ソフトウェアに潜む脆弱性を特定し，セキュリティ要件が満たされているかを検証するプロセスであり[1][2]，ソフトウェア開発ライフサイクルの各フェーズに適用されている[1][3][4].

様々なセキュリティテスト手法が開発・実用化されており，それらは静的手法と動的手法に大別される．静的手法はプログラムの実行を伴わない検証手法であり，要件定義書・仕様書などに対して行われる形式的手法やソースコードに対して行われるデータフロー解析などがある[1][5][6].一方，動的手法は実際にプログラムを実行させて検証する手法であり，ペネトレーションテストやファジングテストなどがある[1][7].

セキュリティテストは，人手によってあるいは自動で実行される．人手によるテストは高い専門性と多大な労力を要する．そのため，自動でテストを実行するセキュリティテストツールの研究が活発になされ，多くのツールが実用化されている．

ソフトウェア開発においてセキュリティテストは広く浸透している．しかしソフトウェアがリリースされたあとに脆弱性が発見される事例は多く，リリース前に完全に脆弱性を取り除くことはできていない．

そこで本稿では，5G ネットワークを脆弱性のない状態で導入することをめざし，自動のセキュリティテストツールの課題について考察する．まず，既存のセキュリティテストツールのしくみを体系的に明らかにする．そのうえで，既存ツールでは発見できなかった脆弱性に着目して，セキュリティツールの問題を特定し，取り組むべき課題を明らかにする．

なお，本稿では，5G ネットワークを脆弱性のない状態で導入するという目的から，通信プロトコルの脆弱性検知に利用可能なセキュリティテストツールを取り上げる．また，5G ネットワーク装置のソフトウェアを自ら開発するのではなく，メーカから製品を調達して受入試験を行うオペレータの立場から，受入試験に適用される動的手法に属するセキュリティテストツールを対象に検討を行う．

## 2. 課題考察の概要

既存のセキュリティテストツールの課題をつぎの 2 段階で考察する．

(1) セキュリティテストツールのしくみの明確化
　セキュリティテストツールのしくみを体系的に明らかにする．まず大局的視点から，既存のセキュリティテストツール全般に共通する 3 つのおもな機能を特定する．つぎに個々のツールがどのようにしてそれらの機能を実現しているかを整理する．

(2) 課題の特定
　既存ツールが検知できなかった脆弱性に着目し，各脆弱性に対して，(1) で整理したしくみのどこに問題があるために検知できなかったか特定する．そのうえで，全体を俯瞰し，脆弱性検知性能を向上させるために効果のある課題を明らかにする．

次章以降で具体的に述べる．

1 NTT ドコモ

## 3. セキュリティテストツールのしくみ

本章では，セキュリティテストツールのしくみについて説明する．まず，既存ツールに共通するおもな機能として，テストデータ生成，事象観測，脆弱性判定の 3 つの機能を特定する．つぎに，テストデータ生成機能に着目し，その実現方法を具体的に整理する．

### 3.1 セキュリティテストツールの機能

本稿で対象とする動的なセキュリティテストツールには，代表的なものとして，既知の脆弱性を検知する脆弱性スキャナや，不正なデータを大量に入力することでクラッシュなどの問題を引き起こして脆弱性を検知するファジングツールなどがある．

それらのツールのサーベイ論文[8][9][10][11][12][13]を調査することによって，セキュリティテストツールにはつぎの 3 つのおもな機能があることを特定した（図 1）．

- テストデータ生成
  試験対象となるソフトウェアに入力するテストデータを生成する機能．
- 事象観測
  テストデータを入力した結果として生じる事象を観測する機能．事象の観測は，試験対象ソフトウェアが実行されるコンピュータ上で，あるいはコンピュータの外部から行われる．観測事象は脆弱性判定の入力として使われるほか，テストデータを改善するためにテストデータ生成のフィードバックとしても用いられる．
- 脆弱性判定
  観測事象から脆弱性の有無や脆弱性タイプを判定する機能．脆弱性判定は，例えば脆弱性を示すレスポンスのシグニチャなど，予め定義された基準にもとづいて行われる．

セキュリティテストツールの課題を明らかにするためには，これら全ての機能について検討する必要があるが，脆弱性を検知するにはそれを発現させるテストデータを入力することが大前提となるため，本稿ではテストデータ生成に着目して検討する．

### 3.2 テストデータの生成方法

・テストデータ生成における課題を特定するため，テストデータ生成のしくみを具体的に明らかにする．そのために，前述のサーベイ論文[8][9][10][11][12][13]で取り上げられているセキュリティテストツールを中心に個々のツールを調査し，テストデータの生成方法を整理する．

調査対象とするツールは，通信プロトコルの脆弱性検知に利用可能な動的セキュリティテストツールとした．動的セキュリティテストツールのなかには，ソースコードの解析結果をテストデータ生成に利用するツールも多数存在する[14][15]．しかしオペレータの立場上，試験対象となるソフトウェアのソースコードを必ずしも入手できるとは限らないため，そのようなツールは除外することとした．

既存ツールでは，つぎのような方法でテストデータを生成している．

- テストデータの変異
  元となるテストデータのビットをランダムに反転させる，あるいは，特定規則に従って変換することで新しいテストデータを生成する[16][17].
- プロトコルに関する情報の利用
  プロトコルのメッセージフォーマット，シーケンスおよび状態遷移などの情報にもとづいてテストデータを生成する[18][19][20]．実現形態として，ツール自体にプロトコルが実装されているものや，機械可読な形式でプロトコルを記述してツールに与えるもの，メッセージのサンプルをツールに与えるものなどがある．また近年では，機械学習によりトラヒックログからメッセージフォーマットを再現する提案もなされている[21].
- 既知の脆弱性・攻撃情報の利用
  既知の脆弱性・攻撃情報にもとづいてテストデータを生成する．実現形態として，既知の脆弱性・攻撃

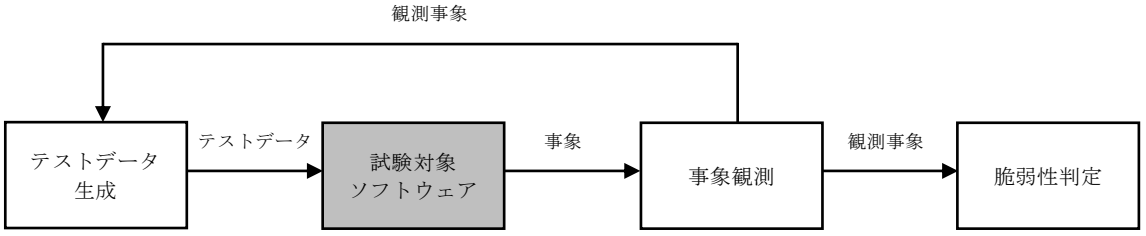


図 1　セキュリティツールのしくみ
Figure 1　A Model of Security Test Tools.

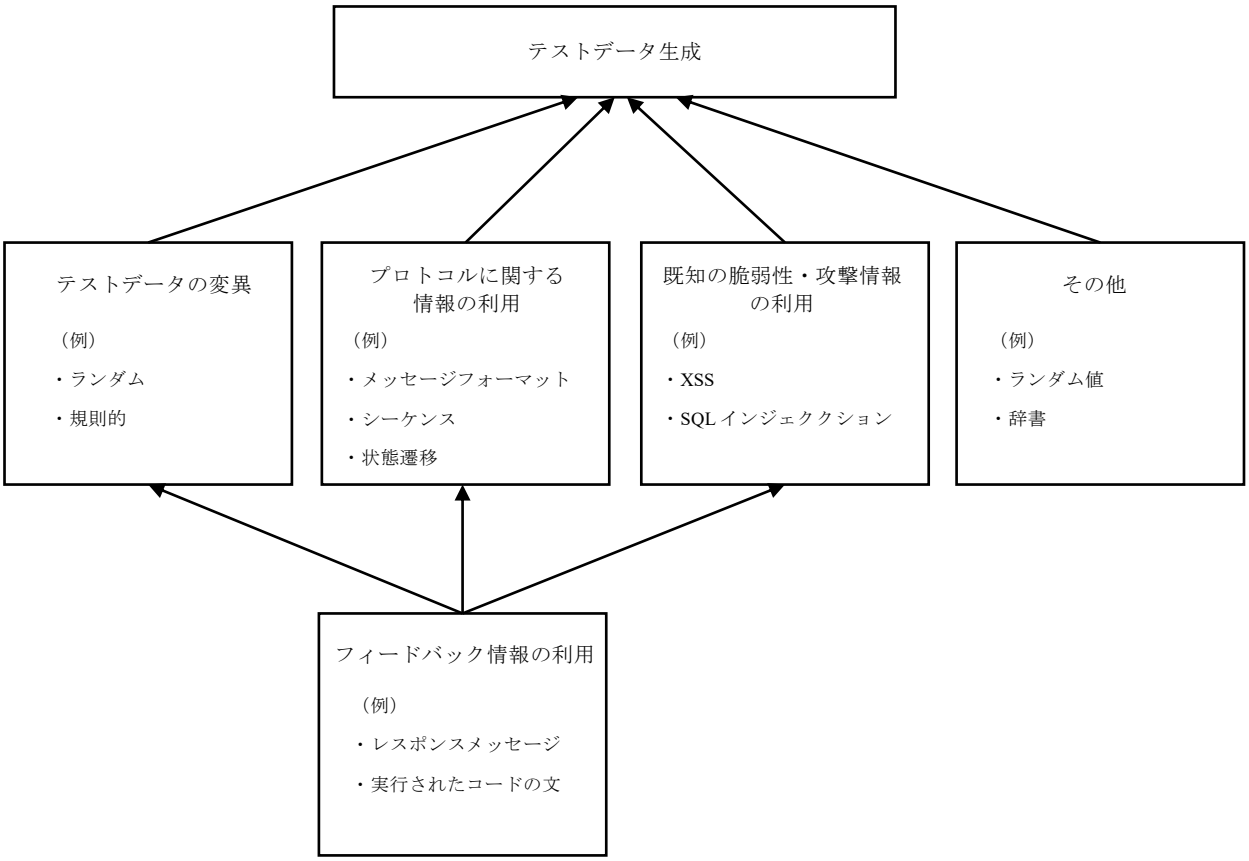


図 2　テストデータ生成のしくみ
Figure 2　A Mechanism of Test Data Generation.

情報にもとづくテストデータを生成するプログラムがツールに実装されている．既知の脆弱性・攻撃情報としては，Common Vulnerabilities and Exposure (CVE)[22], Common Weakness Enumeration (CWE)[23]，OWASP Top 10[24]などが代表的である．

セキュリティテストツールが対応する脆弱性・攻撃は非常に幅広い．例えば，Web アプリケーションスキャナ[25]では，Content Security Policy（CSP）や Cookie などのセキュリティに関係する HTTP ヘッダの検証，コード／コマンド／SQL インジェクション，Cross-Site Scripting（XSS），パストラバーサル，オープン・リダイレクトのほか，多数の脆弱性・攻撃に対応している．

- フィードバック情報の利用
  テストデータ入力によって得られたフィードバック情報を利用して新しいテストデータを生成する．例えば，ファジングツール[16]では，テストデータ入力によって実行されたコード分岐の情報を取得し，コードカバレッジを増加させたテストデータを変異させて新しいテストデータを生成する．Web アプリケーション脆弱性スキャナ[25][26]では，HTTP レスポンスによって返される HTML コンテンツを解析してリンク先の URL や入力フォームを抽出し，テストケースの生成に利用している．
- その他
  その他のしくみとして以下のようなものがある．
  ランダム値：
  テストデータの値をランダムに生成する．
  辞書：
  経験的に脆弱性を発現させやすいことが知られている値（0, -1, MAX_INT など）を用いてテストデータを生成する[27].

図 2 にテストデータ生成のしくみを示す.

## 4. 課題考察

本章では，既存のセキュリティテストツールの課題について考察する．その方法として，実際にセキュリティテストツールにかけられた市中のソフトウェアにおいて，そのあと発見された脆弱性，すなわち，既存ツールでは検知できなかった脆弱性を調査する．それらの脆弱性を前章で整理したテストデータ生成のしくみにマッピングすることで既存ツールの問題の所在を特定し，課題を明らかにする．

本稿では，そのようなソフトウェアとツールの組み合

わせの一例として，Apache HTTP Server[28]とhonggfuzz[17]を取り上げる．honggfuzzは，カバレッジベースのファジングツールである．ツールにあらかじめ設定したメッセージサンプルを最初のテストデータとして試験対象ソフトウェアに入力する．テストデータ入力よって実行されたコードの文および分岐情報を取得し，コードカバレッジを増加させたテストデータを変異させて新たなテストデータを生成する．

honggfuzzのサイトから，Apache HTTP Serverのファジングテストにhonggfuzzが用いられたことが分かる．ただし，テストはApache HTTP Serverを提供するApacheプロジェクトとは無関係に実施されたものと推測される．

honggfuzzが発見できなかったApache HTTP Serverの脆弱性は，マイター社の提供するCommon Vulnerabilities and Exposures（CVE）リスト[22]から，honggfuzzが発見した脆弱性が存在するApache HTTP Serverのバージョンを考慮して抽出した．ただし，honggfuzzはネットワーク経由で試験を行うツールであるため，ローカル環境からのみ攻撃可能なCVEは除外した．

表1に抽出した脆弱性，脆弱性タイプおよび脆弱性を発現させるための入力を示す．脆弱性タイプおよび脆弱性を発現させるための入力は，CVEの記述から読み取れる範囲で記載した．

テストデータ生成における問題を特定するため，表1のCVEを前章で整理したテストデータ生成のしくみにマッピングしたものを図3に示す．例えば，図中の（i）は，テストデータの変異方法を改善することで検知できる可能性のあるCVEを示している．複数の問題にかかわるCVEは複数箇所に記載し，CVEの右側に「*」印を付けている．なお，CVE-2017-7679およびCVE-2018-1312は，図3に記載していない．それぞれ，CVEの記述内容から問題の所在を判断できないこと，およびテストデータ生成の問題ではないことが理由である．

図3から，テストデータ生成における課題はそのしくみ上に全体的に存在していることが分かる．以下では，各問題について考察し，取り組むべき課題を挙げる．

- テストデータ変異の問題
  CVE-2017-7668，CVE-2018-1333，CVE- 2019-0196，CVE- 2019-10082の脆弱性は，ファズの入力によって発現する．ファズの生成方法については活発に研究

表1　既存のセキュリティテストツール（honggfuzz）で発見できなかった脆弱性
Table 1　Vulnerabilities which the Existing Security Test Tool (honggfuzz) could not detect.

| CVE | 脆弱性タイプ | 脆弱性を発現させる入力 |
|---|---|---|
| CVE-2017-7668 | Buffer Over-read | 細工したリクエストの送信 |
| CVE-2017-7679 | Buffer Over-read | 悪意のあるContent-Typeヘッダの送信 |
| CVE-2017-9798 | User After Free | OPTIONメソッドの送信 |
| CVE-2017-15710 | OOB Write | 2文字未満の値をもつAccept-Languageヘッダの送信 |
| CVE-2017-15715 | アクセス制御の回避 | ファイル名に改行文字を挿入したリクエストの送信 |
| CVE-2018-1283 | 不適切な入力検証 | Sessionヘッダの送信 |
| CVE-2018-1312 | 不適切な認証 | ダイジェスト認証で保護されたコンテツへのリクエスト送信 |
| CVE-2018-1333 | リソース浪費 | 細工したリクエストの送信 |
| CVE-2018-11763 | リソース占有 | 大きなSETTINGSフレームの送信 |
| CVE-2018-17189 | リソース浪費 | 緩慢なリクエストの送信 |
| CVE-2018-17199 | 期限切れセッションが利用可能 | 期限切れのCookieヘッダの送信 |
| CVE-2019-0196 | Use After Free | ファズ入力により発生 |
| CVE-2019-0220 | コンポーネント間の処理の一貫性欠如 | URLに連続して「/」を挿入したリクエストの送信 |
| CVE-2019-0217 | 認証回避 | 競合状態下で有効なクレデンシャルを使って別のユーザ名でダイジェスト認証メッセージを送信 |
| CVE-2019-10081 | OOB Write | 非常に速いPush送信（リクエスト頻度，レスポンスサイズに依存） |
| CVE-2019-10082 | Use After Free | コネクションシャットダウン中のファズ入力 |
| CVE-2019-10092 | XSS | XSS攻撃により発生 |
| CVE-2019-10098 | Open Redirect | URLに改行コードを挿入したリクエストの送信 |

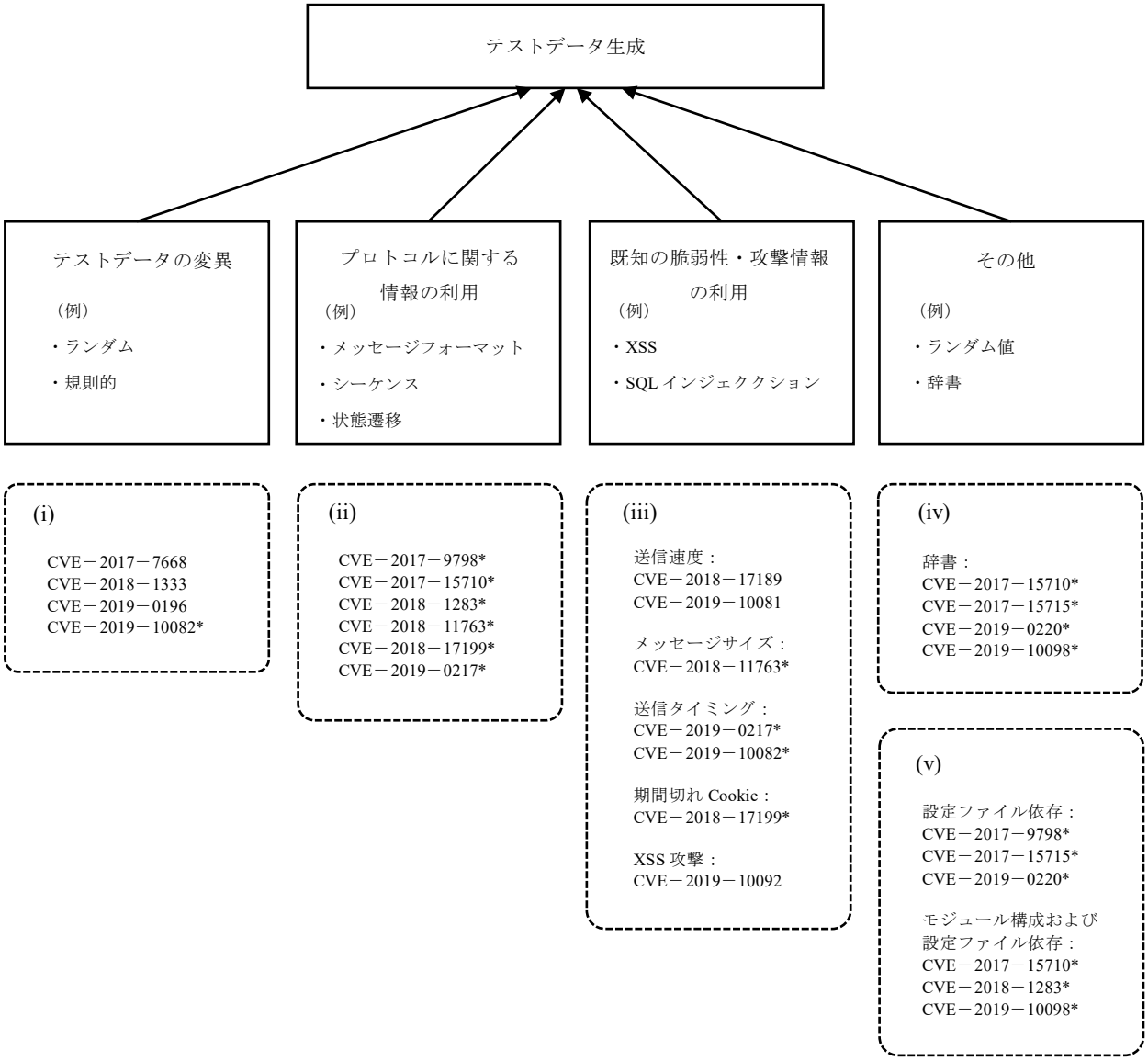


図 3　既存のセキュリティテストツール（honggfuzz）では発見できなかった CVE の
テストデータ生成のしくみ上へのマッピング

Figure 3　Mapping of CVEs which the Existing Security Test Tool (honggfuzz) could not detect on the Mechanism of Test Data Generation.

がなされているが，決定的な解決方法はまだ見つかっていない．本稿の目的を達成するうえでも取り組むべき課題の一つと考えられる．

- プロトコル情報の利用に関する問題
CVE-2017-9798，CVE-2017- 15710，CVE-2018-1283，CVE-2018-11763， CVE-2018-17199，CVE-2019-0271 の脆弱性を発現させるためには，正しいメッセージフォーマットのテストデータを入力する必要がある．本問題は，他の問題との複合条件になっている．従って，プロトコルに準拠したテストデータを生成することは，これらの脆弱性を発現させるための必要条件であると考えられる．

ほとんどのプロトコルの標準仕様は自然言語で記述されている．プロトコルに準拠したテストデータを生成するためには，自然言語で記述された仕様を機械可読な形式に変換する必要がある．人手で変換する場合，多大な労力を要する．

5G ネットワークに関して言えば，コアネットワークの仕様は自然言語の仕様と機械可読なインタフェース仕様の両方が規定されている[29]．一方，無線アクセスネットワークの仕様は自然言語による仕様のみ規定さている．

コアネットワークのインタフェース仕様では，メッセージフォーマットのみが規定されており，シー

ケンスや状態遷移の情報は含まれていない．メッセージフォーマットのみにもとづいてテストデータを生成したので十分か検証することが課題として挙げられる．

一方，無線ネットワークに関しては，自然言語で記述された仕様を機械可読な形式に変換する作業を自動化・省力化することが課題になると考えられる．

- 脆弱性・攻撃情報の利用に関する問題
  本問題に分類した CVE の脆弱性を発現させる方法は多様である．例えば，CVE-2018-17189，CVE-2019-10081 の脆弱性はテストデータの入力速度に依存して発現する．CVE-2018-11763 の脆弱性は大きなサイズのテストデータの入力によって発現する．また，CVE-2019-0217，CVE-2019-10082 の脆弱性は競合状態を引き起こすテストデータの入力によって発現する．

  honggfuzz は既知の脆弱性・攻撃情報を利用していないため，本問題に分類される脆弱性を検知できないのは自然である．既知の脆弱性・攻撃情報を利用するセキュリティテストツールを用いることで，これらの脆弱性を検知できる可能性がある．しかし，そのようなセキュリティテストツールは，試験対象ソフトウェアにおいて既知の脆弱性，あるいは，ソフトウェア毎の違いによる影響が小さく共通的な検知方法を適用可能な未知の脆弱性がおもな検知対象となっている．一方で，honggfuzz のような既知の脆弱性・攻撃情報を利用しないファジングツールは，メモリ不正使用に関する脆弱性の検知がおもな対象となっている．これらのセキュリティテストツールのギャップを埋め，検知可能な脆弱性の範囲を広げることが取り組むべき課題として挙げられる．
- 辞書の問題
  CVE-2017-15710，CVE-2017-15715，CVE-2019-0220 および CVE-2019-E10098 の脆弱性は，改行やスラッシュなどの特定の文字を含む値を設定したテストデータの入力によって発現する．しかし同時に，これらの CVE は Apache HTTP Server の設定ファイルやモジュール構成が特定の条件においてのみ発生するものである．従って，つぎに述べる設定ファイルおよびモジュール構成にかかわる問題の方が本質的な問題と考えられる．
- 設定ファイル・モジュール構成に関する問題
  CVE-2017-9798, CVE-2017-15710, CVE-2017-15715, CVE-2018-1283, CVE-2018-17199, CVE-2019-0220, CVE-2019-10081, CVE-2019-10098 の脆弱性は，Apache HTTP Server の設定ファイルやモジュール構成が特定の設定の場合にのみ発生する．それらの脆弱性を検知するためには，設定条件を変更しながらテストを実施する必要がある．しかし各設定条件に対して単純にテストを繰り返す方法ではテスト量が膨大になる．著者等の知る限り，設定条件とテストデータの関係に着目した研究事例は少なく[30]，設定条件の変更に合わせて効果的に脆弱性を発現させるテストデータの生成が課題として挙げられる．

## 5. まとめ

本稿では，5G ネットワークを脆弱性のない状態で導入することをめざし，通信プロトコルの脆弱性検知に適用可能な動的セキュリティテストツールを対象として，テストデータ生成における課題を明確にした．今後の検討としてつぎが挙げられる．

- 本稿では，既存のセキュリティテストツールでは検知できない脆弱性を調査するため，一つのセキュリティテストツールのみを対象とした．特徴の異なる複数のセキュリティテストツールについて調査し，課題抽出の確度を高める．
- セキュリティテストツールのおもな機能である事象観測および脆弱性判断についても，本稿と同様の検討を行い，セキュリティテストツール全体の課題を明らかにする．

## 参考文献


[1] Felderer, M., Buchler, M., and Johns, M., et al.: Security Testing: A Survey, Advances in Computers, Vol.101, pp.1-51 (2016) .情報処理学会: 情報処理学会論文誌(IPSJ Journal)原稿執筆案内，情報処理学会(オンライン)，入手先 <https://www.ipsj.or.jp/journal/submit/ronbun_j_prms.html> (参照 2020-03-01).

[2] Wikipedia: Security testing (online), available from <https://en.wikipedia.org/wiki/Security_testing> (accessed 2020-05-16).

[3] 情報処理推進機構：ファジング活用の手引き（オンライン），入手先 <https://www.ipa.go.jp/security/vuln/fuzzing.html#001> （参照 2020-05-16）.

[4] Microsoft: Microsoft Security Development Lifecycle (SDL), Microsoft Security Engineering (online), available from <https://www.microsoft.com/en-us/securityengineering/sdl> (accessed 2020-05-16).

[5] Wikipedia: Static program analysis (online), available from <https://en.wikipedia.org/wiki/Static_program_analysis> (accessed 2020-05-16).

[6] 三菱総合研究所：形式手法とは？，ディペンダブル・システムのための形式手法の実践ポータル（オンライン），入手先 <http://formal.mri.co.jp/outline/>（参照 2020-05-16).

[7] 重要生活機器連携セキュリティ協議会：IoT セキュリティ評価検証ガイドライン Rev1.0（オンライン），入手先 <https://www.ccds.or.jp/public/document/other/guidelines/CCDS_IoTセキュリティ評価検証ガイドライン_rev1.0.pdf> （参照 2020-05-16）.

[8] Curphey, M. and Araujo, R.: Web Application Security Assessment Tools, IEEE Security & Privacy, Vol.4, No.4, pp.32-41 (2006).

[9] Bau, J., Bursztein, E., Gupta, D., et al.: State of the Art: Automated Black-box Web Application Vulnerability Testing, Proc. IEEE

Symposium on Security and Privacy, pp.332-345 (2010).
[10] Wang, B., Liu, L., Zhang, J., et al.: Research on Web Application Security Vulnerability Scanning Technology, Proc. IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), pp.1524-1528 (2019).
[11] Li, J., Zhao, B. and Zhang, C.: Fuzzing: a survey, Cybersecurity, Vol.1, No.6 (2018).
[12] Liang, H., Pei, X., Jia, X., et al. : Fuzzing: State of the Art, IEEE Trans. on Reliability, Vol.67, No.3, pp.1199-1218 (2018).
[13] Manes, J.M. V., Han, H., Han, C., et al.: The Art, Science, and Engineering of Fuzzing: A Survey, IEEE Trans. on Software Engineering (Early Access).
[14] Stephens, N., Grosen, J., Salls, C., et al.: Driller: Augmenting Fuzzing Through Selective Symbolic Execution, Proc. Network and Distributed System Security Symposium (NDSS) (2016).
[15] Yun, I., Lee, S., Xu M., et al.: QSYM : A Practical Concolic Execution Engine Tailored for Hybrid Fuzzing, Proc. USENIX Security Symposium (2018).
[16] American Fuzzy Lop (online), available from <https://lcamtuf.coredump.cx/afl/> (accessed 2020-05-16).
[17] honggfuzz (online), available from <https://github.com/google/honggfuzz> (accessed 2020-05-16).
[18] Banks, G., Cova, M., Felmetsger, V., et al.: SNOOZE: Toward a Stateful NetwOrk prOtocol fuzZEr, Proc. International Conference on Information Security, pp.343-358 (2006).
[19] Somorovsky, J.: Systematic Fuzzing and Testing of TLS Libraries, Proc. ACM Conference on Computer and Communications Security (CCS), pp.1492-1504 (2016).
[20] Atlidakis, V., Godefroid, P. and Polishchuk, M.: RESTler: Stateful REST API Fuzzing, Proc. International Conference on Software Engineering (ICSE), pp.748-758, (2019).
[21] Hu, Z., Shi, J., Huang, Y., et al.: GANFuzz: a GAN-based industrial network protocol fuzzing framework, Proc. ACM International Conference on Computing Frontiers (2018).
[22] MITRE: Common Vulnerabilities and Exposures (online), available from <https://cve.mitre.org> (accessed 2020-05-16).
[23] MITRE: Common Weakness Enumeration (online), available from <https://cwe.mitre.org> (accessed 2020-05-16).
[24] OWASP Foundation: OWASP TOP Ten (online), available from <https://owasp.org/www-project-top-ten/> (accessed 2020-05-16).
[25] OWASP Foundation: OWASP ZAP (online) available from <https://owasp.org/www-project-zap/> (accessed 2020-05-16).
[26] PORTSWIGGER: Burp Suite (online), available from <https://portswigger.net/burp> (accessed 2020-05-16).
[27] ALF Technical Whitepaper (online), available from <https://lcamtuf.coredump.cx/afl/technical_details.txt> (accessed 2020-05-16).
[28] Apache Software Fouccation: APACHE HTTP SERVER PROJECT (online), available from <https://httpd.apache.org> (accessed 2020-05-16).
[29] 3GPP: TS 29.501 Technical Specification Group Core Network and Terminals; 5G System; Principles and Guidelines for Services Definition (2019) .
[30] Dai, H., Murphy, C. and Kaiser, G.: CONFU: Configuration Fuzzing Testing Framework for Software Vulnerability Detection, International Journal of Secure Software Engineering (IJSSE), Vol.1, No.3, pp.41-55 (2010).