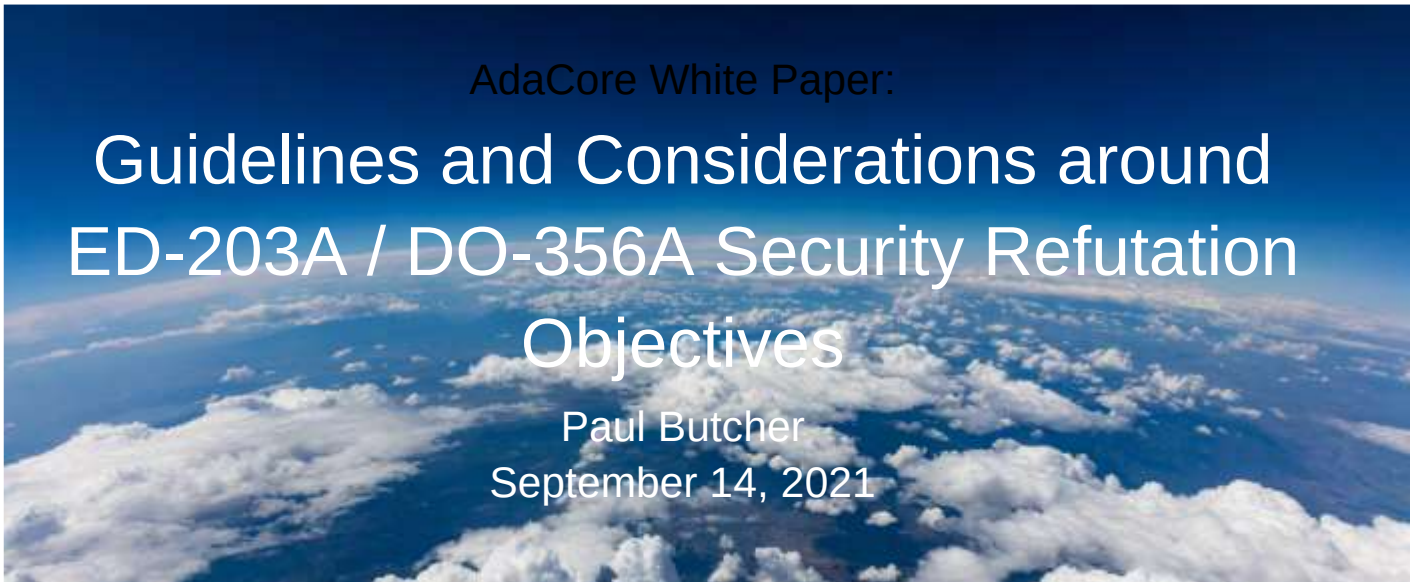
AdaCore



AdaCore White Paper:

Guidelines and Considerations around ED-203A / DO-356A Security Refutation Objectives

Paul Butcher

September 14, 2021

AdaCore ホワイト ペーパー:

ED-203A / DO-356A セキュリティ反証(Refutation)オブジェクティブに関するガイドラインと考慮事項

Paul Butcher (ポール・ブッチャー)2021 年 9 月 14 日

※本資料は、AdaCore の CASE Study(**Guidelines and Considerations Around ED-203A / DO-356A Security Refutation Objectives**)を意訳したものです。正確な内容については、原文を参照して下さい。
https://www.adacore.com/papers/guidelines-and-considerations-around-ed-203a-do-356a-security-refutation-objectives

内容

1. はじめに

このレポートは、「High-Integrity, Complex, Large, Software and Electronic Systems (HICLASS)」プロジェクトの一環で作成されました。HICLASS は、ソフトウェア依存度が高く、安全かつサイバーセキュリティに対抗できるシステムを提供するために創設されました。HICLASS は、英国の航空宇宙サプライチェーン全体を通じて新しい技術と最善の手法を牽引する戦略的取り組みであり、英国で向こう数十年にわたって成長すると予想されている航空機とアビオニクス市場向けのシステムを、低コストで開発ができるようになります。HICLASS プロジェクトは、民間航空宇宙設計製造における英国の競争力を維持、向上させるため、政府と産業界が共同で出資する航空宇宙テクノロジ研究所(ATI)プログラムの支援を受けています。ATI、ビジネス、エネルギー、産業戦略省 (BEIS)、Innovate UK のパートナーシップを通じて提供されるこのプログラムは、テクノロジ、機能、サプライチェーンの課題に取り組んでいます。ロールスロイスコントロールシステムズが主導する 3,200 万ポンドの投資プログラムは、英国の民間航空宇宙部門に重点を置いていますが、防衛科学技術研究所(DSTL)とも直接関わっています。

HICLASS は、複数の作業部会 (WP) に分かれています。WP1.2 は、航空宇宙セキュリティ規格の解析、規格の完成度に関する差分(maturity gap)を特定し、セキュリティオブジェクティブに適合する最善の手法を共有することを目的とした共同作業グループとして設立されました。HICLASS ドキュメント D1.2.4TH 「航空セキュリティおよびサイバー セキュリティ プロセスの識別に関するレポート」は、「セキュリティ プロセス規格解析」の最初の成果物であり、このレポートで、反証(Refutation)テストに関するガイダンスの必要性が明らかになりました。

このレポートは、欧州民間航空機器機構 (EUROCAE) ED-203A「耐空性セキュリティ手法と考慮事項(Airworthiness Security Methods and Considerations)」[6] で説明されているセキュリティ反証(Refutation)オブジェクティブに関するガイドラインと考慮事項を解説することを目的としています。ED-203A の序文では、ED-203A は技術的には RTCA DO-356A [4] と同一で、ED-202A [3] および RTCA DO-326A [5] にも当てはまります。さらに、ED-202A プロセス仕様と ED-203A の手法と考慮事項(ED-202A process specification and ED-203A methods and considerations)を参照していますが、ガイドラインと考慮事項に関しては、DO-326A と DO-356A にも適用可能です。

反証(Refutation)は ED-203A / DO-356A で次のように説明されています。

反証(Refutation)は、解析や要件を超えた独立した保証手段(assurance activities)として機能します。徹底的なテストに代わるものとして、反証(Refutation)は、許容可能な範囲で、予期しない振る舞いが抑止されているという証拠を提示するために使用可能です。
注: 反証(Refutation)は、状況によってはセキュリティ評価とも呼ばれます。

耐空性セキュリティプロセスの観点から見た「反証（Refutation）」の目的は、攻撃可能な脆弱性の問題に反論することで、単にシステムが安全であると宣言するのとは対照的に、否定的な表現にしたのは意図的です。

これは、関連する活動の否定的なテストを重視し、反証（Refutation）テスト フェーズをセキュリティ検証テストと区別するためです。この違いは微妙ですが、理解することが重要です。セキュリティ検証活動（肯定的テスト）は、セキュリティ要求事項が満たされていることを論証すること、あるいは、場合によっては、証明することに重点を置きます。否定的なテスト活動では、セキュリティ要求事項が満たされているという仮説に反証することに焦点を当てますが、同時に「許容可能な範囲で、予期しない振る舞いが抑止されている」という保証も得ています。

反証（Refutation）は、標準的なセキュリティ検証テストとは多くの点で異なります。ファズテストやペネトレーションテストなどの一部の活動において、要件で動作が定義されていない場合、システムが想定外での状況下でのシステム動作の観察に重点を置いています。反証（Refutation）活動の目的は、システムが安全ではない状態に遷移する（一般的には、システムのセキュリティを保証する最小限の不変条件に違反する）状況を特定することです。形式要件ベースのテスト手法（formal standard requirements based testing methods）を使用して、この状況を特定すると、想定外の事態が発生する可能性が高く、代替策は、システムの特定の箇所をテスト対象にして、要件に基づく不要な偏りを排除したシナリオを生成することです。

さらに、反証（Refutation）の目的を理解するためには、「脆弱性」という用語の意味と、「ソフトウェアのバグ」と「脆弱性」の違いと類似点を理解することが重要です。

ED-202A では次のように定義しています。

「脆弱性：システムのセキュリティ手順、設計、実装、または内部管理における欠陥または弱点であり、それが攻撃されると（意図せずに攻撃される、あるいは意図的に攻撃される）、セキュリティ侵害やシステムのセキュリティポリシー違反につながる可能性がある。」[3][5]

ここで、ソフトウェアのバグは、「利用される可能性がある」（つまり、攻撃される可能性がある）こと、および「セキュリティ侵害またはシステムのセキュリティ ポリシー違反につながる可能性がある」ことが実証されている場合にのみ、脆弱性となります。ただし、ED-203A には、次のことも記載されています。

脆弱性として判断される前に、バグまたは欠陥として認識されている場合があります。脆弱性が何年も欠陥として認識されていたが、潜在的な攻撃に気付かなかったという事例があります。また、脆弱性が「修正された」（既知の攻撃の緩和または防止によって）と判断された場合でも、数年後に新たな攻撃によってまだ存在していたこともあります。」[6]

これは ED-202A の定義と矛盾しています。なぜなら、既知の攻撃手法が不明で、結果としてセキュリティ侵害やシステムのセキュリティ ポリシー違反につながる経路が不明なソフトウェアバグも、脆弱性（あるいは、少なくとも「潜在的な脆弱性」）としてみなすべきです。

この警告は、ソフトウェアバグの分類や特定された脆弱性を緩和する潜在的な安全上のリスクを明確に示しています。このことは、セキュリティ脆弱性特定フェーズ 1 で、多くの欠陥が特定されるようにシステムを確実に運用することの重要性を示しています。次に、フェーズ 2 では、欠陥を脆弱性（関連する安全上の危険性に基づいて計算された等級）、または非脆弱性ソフトウェアバグ（将来、この分類が変更される可能性が低い場合）に分類します。また、ソフトウェアバグを脆弱性（すなわち攻撃可能）に分類する論拠は、ソフトウェアバグを攻撃不可能に分類する論拠と同様に強固である必要があります。

もう一つの考慮事項は、ED-202A では、ソフトウェアバグは「意図せずに発生する」可能性がある場合、脆弱性として分類されると規定されています。ソフトウェアの変更によって、バグが無意識のうちに非脆弱性から脆弱性に移行する可能性があり、非脆弱性ソフトウェアバグに共通するもう一つの問題を浮き彫りにしています。

多くの反証（Refutation）テスト機能、ならびにテストチームが採用する手法によって、特定された欠陥が攻撃可能かを分類できるかは、その活動によって決まります。特定された欠陥が攻撃可能であると判断されても、その欠陥が自動的に脆弱性になるわけではありません。定義に示しているように、脆弱性として分類するには、欠陥が「(意図せずに攻撃される、意図的に攻撃される) セキュリティ侵害またはシステムのセキュリティ ポリシー違反につながる必要があります。

ペネトレーションテストなどの活動は、攻撃経路がわかっているシステムへのエントリを対象としています。特定された異常は、攻撃される可能性が非常に高くなります。比較すると、形式証明の静的アナライザーなどのツールは、欠陥を発見するシナリオを確認できます。さらに解析すると、攻撃者はソフトウェアバグを攻撃できる既知の手法がないと判断する可能性があります。例えば、アナライザーは、誤ったデータ型を使用したために発生する可能性のあるオーバーフロー例外を特定できますが、システムバンダリの入力変換（sanitizer）が原因で、エラーが発生するデータを生成できない場合があります。

ファジングは、ファジングキャンペーンの範囲に応じて、攻撃不可能な欠陥と攻撃可能な欠陥の両方を特定する必要があります。ファジングテストハーネスを構成して、ファジングキャンペーンがシステムへのエントリ ポイント (攻撃者がエントリにアクセスできるポイント) をターゲットにできる場合、特定された異常は、すべて攻撃可能と見なす必要があります。キャンペーンの範囲が、システムの個々の部分を実行するために多数のテストが必要になるように設定されている場合は、特定された異常は攻撃可能である場合とそうでない場合があります。

ソフトウェアバグが実際に脆弱性であるかを判断することは、現在のソフトウェアライフサイクルがどの段階にあるかによって異なります。ソフトウェアバグは、ライフサイクルの後半で発見されるほど修正に手間がかかり、その場合はソフトウェアバグを修正して再テストすることが最善の解決策となります。

アプリケーションが動作中で、発見されたが脆弱性を確認できないソフトウェアバグを修正することは、困難な場合があります。このシナリオでは、既知の欠陥が脆弱性ではなく、攻撃ができないため、ソフトウェアバグが航空機の安全上の危険につながることはないという証拠を示す必要があります。問題は、反証（Refutation）活動が既知の攻撃手法のみに焦点を当てていることで、すべてのバグが特定され、脆弱性として分類されていることを論証することが難しい場合があります。いずれにしても、反証（Refutation）中に発見されたすべての欠陥と異常が修正されるか、十分に緩和された場合、論理的にソフトウェア システムに脆弱性が含まれていないという強力な論証となります。

耐空性セキュリティ プロセスのコンテキストでは、ソフトウェア開発ライフサイクル内の反証（Refutation）フェーズの包括的なオブジェクティブは、システムがセキュリティ保護されている（したがって安全である）ことを保証することです。反証（Refutation）テスト活動のオブジェクティブは、システム内の脆弱性を特定し、実装されたセキュリティ対策の堅牢性をテストすることです。

システムが安全であるという論証に対する信頼性を証明するには、特定されたすべての脆弱性を次のいずれかの方法で緩和（受け入れる）される必要があります。

- 修正（脆弱性がソフトウェアのバグに関連していると想定）。
- 脆弱性ファイル内に記録され、脆弱性の攻撃に関連する安全上の危険に見合ったセキュリティ対策または一連のセキュリティ対策によって保護。
- 脆弱性ファイル内に記録され、航空機プラットフォーム全体のより高いレベルのメカニズムを通じて緩和。
- 脆弱性ファイル内に記録し、脆弱性が攻撃される計算上のリスクが十分に低いことを承認。

脆弱性が特定できない場合は、次のいずれかの手法となります。

- 反証（Refutation）ではバグや異常は特定できない。
- バグや異常が特定され、特定されたバグや異常が攻撃されるリスクは許容できることを論証します。
- バグまたは異常が特定され、発見されたバグ、または異常の 一つ以上が攻撃可能であることを実証します（意図的または非意図的）。ただし、攻撃されたバグや異常によってセキュリティ侵害やシステムのセキュリティ ポリシー違反が発生するリスクは許容できると十分に論証します。

さらに、反証（Refutation）フェーズの計画や実行が不十分だと、当然のことながら、バグや異常をすべて特定できない可能性があります。このシナリオでは、システムに脆弱性が存在する可能性がありますが、まだ発見できていない場合があります。

ED-203A / DO-356A には、反証（Refutation）には次のテスト/解析活動が含まれると規定されています。

- セキュリティ ペネトレーションテスト
- ファジング
- 静的コード解析
- 動的コード解析
- 形式証明

この活動の定義は付属文書のセクションに記載されています（関連する付録が利用できる場合）。ただし、レポートでは、次の定義を使用します。

セキュリティペネトレーションテスト: 特定されたセキュリティ資産の状態を読み取ったり変更したりすることを目的として、システムの脆弱性を攻撃するテストメカニズムです。

ファジング: 自動化されたテストケース入力生成およびシステム異常検出メカニズム。ファジング機能では通常、テストケース入力の開始コーパスのミューテーションが行われ、システムが異常な動作を観察している間にテストケースをさらに生成することを目的としています。ファジングの目的は、システムを安全でない状態に移行させる可能性のあるシステム入力を特定することです。

静的コード解析: システムのソースコードレベルでのセマンティック解析。安全ではない、セキュアではないと考えられるコード構造を識別することを目的として実行されます。

動的コード解析: システム実行中のシステムの動作の解析。動的コード解析の例としては、プロセッサに対して実行される安全ではない、セキュアではない命令呼び出しシーケンスの監視（バッファオーバーフローの検出など）が挙げられます。動的コード解析は、実行時の制約チェック、または追加のコンパイル中に追加されたコードインストルメンテーションを利用して、メモリを破壊するバグを検出するツールを介して、プログラミング内で実行できます。

形式証明: ソフトウェア設計時に作成され、システムの意図された動作を記述することを目的として、数学的な証明チェッカーによって公式(公理、仮定、規則、推論 – 契約（contracts）およびアサーションとも呼ばれる)を静的に検証することができ、曖昧性がなくプログラムによって検証可能な有限数式のシーケンス。
システムが「契約による設計（"Design by Contract"）」方法論で開発され、ソフトウェア実装内のセキュリティ要件を指定するために形式手法が使用されている場合、静的検証ツールを使用して、システムで実行される情報フローのセキュリティの安全性を保証できます。さらに、自動化された形式証明ツールを使用して、実行時エラー（サービス妨害（DoS）攻撃につながる）が存在しないということを保証できます。

上記は、反証（Refutation）活動を網羅したのではなく、それぞれの分野を組み合わせたり、重複させたりすることも有益です。例えば、一部のファジング実装では、動的コード解析を使用して安全ではないシステムの動作を検証します。

2. 目的

HICLASS レポート 1.2.4TH「航空セキュリティおよびサイバーセキュリティプロセスの識別に関するレポート」で特定された次の問題に関するガイドラインの提供と計画の考慮事項の検討:
15 号（Issue 15）

「ED-203A には、「反証（Refutation）テスト」の目的と活動が含まれています。ただし、適切な反証（Refutation）/ペネトレーションテスト（量と質の面で）を決定することは実際には難しく、ガイダンスは提供されていません。
例えば、どの程度で十分なのか？ 何をテストするのか（例: モデルや自動生成コード）？ プログラミング言語の選択（例: C、Ada）は、テストにどのような影響を与えるのか？ ここでは追加のガイダンスが役立ちます。」

ただし、考慮事項が広範囲に及ぶ場合やプロジェクトに固有の場合など、一部の考慮事項に関連する具体的なガイドラインがない場合がありますこのような場合、附属文書の項目は、読者が各プロジェクト固有の反証（Refutation）計画活動を結論付けることができるように、確実な活動特有の検討事項を準備する必要があります。考慮事項が特定の活動に適用されないこともあり、十分な説明が用意されている場合に容認されます。さらに、このレポートには記載がありませんが、反証（Refutation）活動に適用可能な追加の活動考慮事項がある場合は、附属文書に記載する必要があります。

3. スコープ

このレポートは、セキュリティ反証（Refutation）テスト活動の計画方法と、ED-203A / DO-356A で定義されたセキュリティ反証（Refutation）オブジェクティブを満たすために考慮すべき事項に焦点を当てています。また、このレポートでは、読者が ED-202A / DO-326A プロセス仕様と ED-203A / DO-356A の方法と考慮事項を十分に理解していることを前提としています。ただし、このレポートの範囲を説明するには、ED-202A / DO-326A と ED-203A/DO-356A 内の基本的な用語と概念のいくつかを繰り返し述べる必要があります。さらに、プロセスを解釈する場合は、プロセスに関連する仮定を説明して明確に必要があります。

ED-203A/DO-356A に記載されているセキュリティ反証（Refutation）の目的は次のとおりです。

- O3.1 反証（Refutation）解析は新たな脆弱性を特定するために行う。[6][4]
- O3.2 反証（Refutation）テストは、セキュリティ環境における脆弱性の露出を評価し、脆弱性評価に異議を唱えるために実行される。[6][4]
- O3.3 反証（Refutation）テスト計画が利用可能である。反証（Refutation）テスト結果には、反証（Refutation）テスト計画と実行されたテストが含まれる。反証（Refutation）テスト結果が解析され、矛盾が正当化され、追跡される。[6][4]

したがって、セキュリティ反証（Refutation）の主な目的は、システム内の脆弱性を特定することです。二次的かつ必須である目的は、特定された各脆弱性の攻撃可能性を評価できるようにすることです。つまり、ソフトウェアのバグ、ハードウェアの欠陥などが攻撃経路の一部となるかを明確にすることです。

3 番目の目的は、次の三つの目的に分けられます。

1. 反証（Refutation）テスト計画は、認証のセキュリティ面に関する計画 (PSecAC) の一部として作成されました。

通常、これは、認証機関からの PSecAC 承認が必要になります。このルールの例外は、入力活動 (航空機セキュリティ範囲定義 (ASSD 出力)、航空機セキュリティ アーキテクチャ (ASAM 出力)、航空機脅威条件など) によって、航空機セキュリティの影響の重大度が「影響なし」と示されている場合、または「軽微」な影響が認証機関と交渉して解消された場合です。いずれの場合も、PSecAC は、計画がセキュリティの規制要件にどのように対処するかを示す必要があり、計画された反証（Refutation）活動の詳細な内訳を含める必要があります。

2. 実行された反証（Refutation）テストが、反証（Refutation）テスト計画に記載されている活動をカバーしていることを確認します。

テストに関係する活動と同様に、その活動を示している必要があります。

- 目的に適合している（宣言されたテストを実行している）、
- 正しく実行された（テストの実行が、十分に管理された環境内で対象となるセキュリティを、正しく評価したと確信できる）、
- 記録され、再現可能である（テストの結果が保存され、同じ動作と結果が観察されるようにテストを再実行できる）。

3. テスト結果が解析され、理解され、該当する場合は偽陽性と偽陰性が文書化されていることを確認します。

反証（Refutation）テストの多くの承認された形式には、脆弱性検出機能に既知の限界があり、偽陰性の結果をもたらすことが予想されます。さらに、静的アナライザーなどのツールは、多くの場合、偽陽性の結果となり、解析アルゴリズムの限界を克服するためにユーザーの操作が必要になります。このような場合、限界を理解して PSecAC 内で文書化し、テスト結果の解析で既知の限界が考慮されるようにするプロセスを採用することが必要です。多くの場合、同等の DO-330/ED-215 ツール認定レベル (TQL) に準拠することで、反証（Refutation）テストツールの限界を理解するための情報を得ることができます。ただし、ED-203A では次のように規定されています。

- 4.2.8「ツールのセキュリティオブジェクティブ」
    - 脆弱性を検出するために使用するツール
    - ツールは ED-215 [8] / DO-330 [1] TQL-5 に従って認定される。
    - 関連するツールとは、静的コード解析など、開発中の製品の脆弱性を検出できない可能性のあるツールで、ED-215 / DO-330 または ED-80 / DO-254 に従って認定される。
    - 反証（Refutation）テスト内で攻撃を実行するためにのみ使用されるツール、スクリプト、およびサポート データは、セキュリティ ツール認定の対象ではない。

- 注：TQL-5 の認定オブジェクティブは、ED-215 / DO-330 のセクション 11.3 に記載されているように、ツール開発者からのツール認定データがなくても認定される場合がある。[6][4]

反証（Refutation）テストツールの保証が関連する TQL オブジェクティブの達成で得られるかにかかわらず、ツールの既知の制限は PSecAC 内に文書化する必要があります。

さらに、セキュリティ反証（Refutation）活動では、ある活動の限界を別の活動の能力によって緩和する階層化アプローチを採用する必要があります。この考え方は、セキュリティ アーキテクチャと対策のために開発する際の考慮事項として ED-203A が提案する「多層防御」の理念にも従っています。ここでは、次のことが重要です。

「多層防御は、さまざまな複雑さを持つ複数の脅威から身を守るための好ましい手段であるために必要です。攻撃には多くの種類があり、未知の攻撃手法に備えるために、脅威から身を守る際には、複数のレイヤーでさまざまな技術的概念を使用する必要があります。」[6][4]。

脆弱性を特定しようとする場合も同様です。一つの反証（Refutation）テスト方法論だけでは十分ではない可能性があります。補完的なセキュリティ反証（Refutation）活動の例として、定理証明では簡単には達成できない、アプリケーションのサブシステムの実行時エラーがないことを確信するためにファズテストを使用します。採用されたセキュリティ反証（Refutation）活動と、それらが互いにどのように補完し合うかについては、セキュリティ反証（Refutation）テスト計画に記述する必要があります。

セキュリティ反証（Refutation）を計画する際には、既知の攻撃に重点を置く活動と、攻撃経路の一部となる可能性のある、そうではない潜在的な脆弱性を探す活動の差異を理解することも重要です。どちらの手法も同じように適用可能であり、両方を採用すれば、より優れたセキュリティアシュアランスを実現できます。

ED-202A / DO-326A および関連文書では、「セキュリティ反証（Refutation）」という用語について明示的に言及または定義されていません。これは、ED-203A / DO-356A で次のように説明されています。

「 ED-203A/DO-356A では、耐空性保証の方法として、保証活動を説明し収集するために 反証（Refutation） という用語を導入しています。この用語は新しいもので、現在の ED-202A / DO-326A のバージョンでは使用されず、他の公開された航空規格でも使用されていません。文書（ED-79A / ARP4754A、ED-12C / DO-178C、ED-80 / DO-254 など）で言及されている反証（Refutation）活動の多くは、安全性/セキュリティアシュアランスオブジェクティブに関連する活動の一環として議論されています。」[6][4]

したがって、反証（Refutation）テストと脆弱性テストは新しく、ED-202A および ED-203A 以前は、耐空性認証における標準的な手法とは見なされていませんでした。そのため、この分野は十分に確立された方法、考慮事項、ガイドラインが欠けており、このレポートでは問題点を解説します。

セキュリティ反証（Refutation）は、耐空性セキュリティ プロセス内の脆弱性特定およびセキュリティ対策評価活動の一部を形成するため、セキュリティアシュアランスにおいて重要な役割を果たしています。ED-202A / DO-326A では、セキュリティアシュアランスは次のように説明されています。

「 セキュリティ対策が意図したとおりに機能し、開発中に導入される可能性のある既知の攻撃可能な脆弱性が最終製品に存在しないことを保証することが必要とされる 」[3][5]。

しかし、認証のセキュリティ対策の計画（Plan for Security Aspects of Certification: PSecAC）の中で、どこに反証計画を記述すべきかを判断するとき、混乱を招く可能性があります。ED-203A / DO-356A の説明には、「反証（Refutation）はセキュリティ評価とも呼ばれます」と記載されています。これは、ED-202A / DO-326A 内では反証（Refutation）が「セキュリティ検証」の範囲内に含まれることを意味しています。セキュリティ検証の目的は、セキュリティの有効性の評価に貢献することです。

「セキュリティ検証の目的は次のとおりです。

- セキュリティの有効性の評価に貢献します（セキュリティに特化）。[ED-202A / DO-326A] [3][5]。

耐空性セキュリティ プロセスでは、セキュリティ有効性の要件は「航空機セキュリティ アーキテクチャと対策」の開発中に導き出され、「予備的な航空機セキュリティ リスク評価」に対して評価されます。セキュリティ有効性の要件は、航空機セキュリティ検証（Aircraft Security Verification (ASV)） への入力となり、この演習のオブジェクティブは次のとおりです。

「脆弱性テスト計画を最終決定し、セキュリティリスク評価における脅威シナリオにおいて、セキュリティの有効性を含め、セキュリティ要件をどの程度カバーしているかを評価する。 」[ED-203A / DO-356A] [3][5]。

多くの場合、セキュリティ要件の検証は、DO-178C [2] に記載されている規格の検証手法に準拠することで実現できます。ただし、これはセキュリティ要件が採用する個々の形式に依存します。ここでの前提は、要件が肯定的 (有限のテストケース) 表記法で表現されているため、規格の検証手法を採用できます。この場合、セキュリティ要件の検証に追加してセキュリティ反証（Refutation）テストは必要ではありません。セキュリティ要件が否定的テストを優先する表記法で表現されている場合、反証（Refutation）テスト活動を使用して要件が満たされている証拠を提供することは可能ですが、どのように達成するかは、このレポートの範囲外です。

さらに、ASV のもう一つのオブジェクティブは次のとおりです。

「脆弱性評価とテストを実施し、セキュリティリスクの結果を解析する。」[ED-203A / DO-356A]

このオブジェクティブは、セキュリティ反証（Refutation）テスト方法論を採用することで達成できます。さらに、セキュリティ反証（Refutation）テストの結果は、航空機脆弱性文書の一部となります。ASV によって達成された航空機セキュリティ検証およびテストの結果と解析も、PSecAC 概要レポート内に文書化する必要があります。

ED-202A / DO-326A では、セキュリティ検証の目的が次のように規定されています。

- システム、ハードウェア、ソフトウェアがセキュリティ要件を満たしていることの検証（ED-79A / ARP 4754A などの規格開発プロセスにおける通常の目的）、および
- セキュリティの有効性の評価に貢献（セキュリティに特化）
- 最終的なセキュリティリスク評価のための脆弱性の特定（セキュリティに特化）。[ED-202A / DO-326A]

耐空性セキュリティ プロセス（Airworthiness Security Processes （AWSP））では、さらに 三つのテスト活動について説明します。

- セキュリティ要件テスト。
- セキュリティ堅牢性テスト。
- 脆弱性テスト。

ED-202A / DO-326A では、セキュリティ要件テストとセキュリティ堅牢性テストについて次のように規定されています。

「異常な入力や状況下でも技術的セキュリティ要件の実装が正しいことを実証する。」[ED-202A / DO-326A]

したがって、最初の 二つのテスト活動は、セキュリティ要件の検証に基づいており、前述のように、特に安全関連の障害状態に関連付けられている航空機ソフトウェアを開発する場合に「標準的な方法」と見なすことができます。ただし、テストの焦点は、安全要件ではなくセキュリティ要件となります。

3 番目の「脆弱性テスト」は、要件に明示的に焦点を当てておらず、適用可能なセキュリティ反証（Refutation）活動を採用することで満たすことができます。

耐空性セキュリティ プロセス（AWSP）仕様では、「脅威条件」の概念を導入することで、航空機搭載システムおよび機器の認証における既存のソフトウェアの考慮事項を拡張しています。「脅威条件」は、次の概念を導入する点で「障害条件」とは異なります。

「 サイバー脅威を伴う意図的な不正の電子的やり取り（IUEI）行為... 」[ED-202A / DO-326A]。

ED-203A では脆弱性は次のように説明されています。

「システムのセキュリティ手順、設計、実装、または内部制御における欠陥または弱点。これらが攻撃されると（意図せず誘発する、意図的に攻撃される）、セキュリティ侵害またはシステムのセキュリティ ポリシー違反につながる可能性があります。」[ED-203A]。

したがって、脅威条件には、意図的または意図せずに引き起こされたシステムの脆弱性の攻撃が含まれ、その攻撃は航空機やその乗員に直接的、または結果的な影響を及ぼす可能性があります。反証（Refutation）でカバーされている保証活動の目的は、システムの堅牢性において許容可能なレベルの信頼性を実証できることを主張することです。この内容で、「堅牢なシステム」とは、脅威条件につながる可能性のある脆弱性を含まないシステムを示します。

耐空性セキュリティプロセス（AWSP）のコンテキストにおける反証（Refutation）活動と、それが認証の取得において果たす重要な役割を理解するには、AWSP の主なオブジェクティブを考慮することが重要です。ED-202A / DO-326A では、第 2 章のオブジェクティブを次のように説明しています。

「耐空性セキュリティプロセス（AWSP）の目的は、許可されていない相互作用を受けた場合でも、航空機が安全に運航できる状態を維持することを確立することです（規制上の耐空性基準を使用）。この目的を達成するために、耐空性保安プロセスは次のことを行います。

- 航空機とそのシステムに対するセキュリティリスクが AWSP によって確立された基準に従って許容可能であることを確立し、
- 耐空性セキュリティリスク評価が完全かつ正確であることを確認します。

[ED-202A / DO-356A]。

反証（Refutation）テストは、「航空機とそのシステムに対するセキュリティ リスクは許容できる」という主張を裏付ける証拠を収集する役割を果たします。さらに、適切な反証（Refutation）セキュリティ テスト プランがある場合にのみ、規制当局は「耐空性セキュリティリスク評価が完全かつ正確である」ことを立証できます。

ED-203A/DO-356A セキュリティ反証（Refutation）オブジェクティブへの適合させるための潜在的な方法となる「反証テスト」の領域に該当するセキュリティテスト分野の数は膨大です。加えて、新規及び既存のテスト能力とテスト方法論は、既存及び新規のソフトウェアの脆弱性の理解が深まるにつれて、急速に進化しています。
ED-203A / DO-356A では、反証（Refutation）には次のテストおよび解析活動が含まれると規定されています。

- セキュリティペネトレーションテスト
- ファジング
- 静的コード解析
- 動的コード解析
- 形式証明

ただし、これはすべてを網羅したのではなく、既存および将来の活動も重要な役割を果たします。さらに、ED-203A は、ED-202A と RTCA の同等物である DO-326A の両方にガイドラインを提供するために制定されましたが、認証取得のために他の方法を利用することができます。

このレポートのメインセクションでは、反証（Refutation）テストに関する考慮事項を特定します。次に、リストされている活動ごとに付属文書に記載されます。これにより、新しい反証（Refutation）活動の付属文書が利用可能になったときに、随時追加して拡張することができます。各附属文書は、その活動との関連性を述べながら、該当する場合には、セキュリティの反証（Refutation）オブジェクティブを達成するためのガイドラインを提供することにより、考慮事項を説明します。

このレポート内のガイドラインでは、偏りを避けるため、特定の市販ツールについては言及しません。また、プログラミング言語は、セキュリティ関連の考慮事項において付加価値がある際に取り上げます。

このレポートでは、ED-202A / DO-326A 耐空性セキュリティ プロセス フレームワークで定義されている「認証のセキュリティに関する計画 (PSecAC)」に適合する反証（Refutation）テストに関する考慮事項を提案します。

システム機能テスト、セキュリティ対策テスト、セキュリティ堅牢性テストなどのその他の活動は対象外となります。

4. 考慮事項

4.1 一般的な考慮事項

4.1.1 視点

既存の確立された耐空性安全プロセスでは、意図的な妨害は考慮されておらず、代わりに、事故につながる可能性のある周辺環境の危険や内部コンポーネントの故障などに重点が置かれています。安全性の「故障状態」からセキュリティの「脅威状態」への移行には、視点の変更と、検証と反証（Refutation）の違いを理解することが必要です。
ED-203A / DO-356A では、次のように述べてこの点を強調しています。

「検証と反証（Refutation）の活動は異なる概念に従うため、別々に実行する必要があります。検証活動は要件ベースですが、反証（Refutation）活動は攻撃者の視点から実行する必要があります。」[ED-203A / DO-356A]。

したがって、標準的な検証ベースのテスト手法と異なる反証（Refutation）テストの重要な点は、反証（Refutation）テストチームがテスト手法とテスト計画を決定する視点です。

セキュリティ要件が満たされていることを証明するために標準的な検証テスト手法を採用する必要がありますが、攻撃者の視点から考えると、攻撃可能な脆弱性を見つけることが可能です。実際には、難しい手法で、成果を測るのはさらに困難です。ただし、重要なのは、攻撃者が不正な電子的操作を実行するために使用した手法を反映した反証（Refutation）活動を実施することです。

4.1.2 セキュリティコーディングスタンダード

ED-203A / DO-356A のオブジェクティブ O2.1 には次のように記載されています。

「セキュリティ対策と資産（COTS を含む）の脆弱性が特定され、安全性への影響が評価されます。」
[ED-203A / DO-356A]

さらに、オブジェクティブには次の注釈がついています。
「脆弱性が、脆弱性として認識される前に「バグ」または欠陥である可能性を考慮する必要があります。」
[ED-203A / DO-356A]。

したがって、ソフトウェアバグは「脆弱性の特定」の一環として特定する必要があります。脆弱性の特定がシステムの新規開発または更新時に行われている場合、ソフトウェアバグは修正する必要があります。ソフトウェアパッチが用意されていない場合、または既存のシステムで遡及的な脆弱性の特定が行われている場合は、特定されたバグまたは欠陥の「攻撃可能性（exploitability）」を評価する必要があります。この評価では、バグや欠陥を解析し、攻撃手段が定義されたシステムセキュリティ領域を越えて攻撃を行うことができるかどうかを判断する必要があります。特定されたすべての脆弱性は、「脆弱性文書」に記述する必要があります。

さらに、ED-203A / DO-356A 実装オブジェクティブ O8.2 には次のように記載されています。
「ソースコードならびにハードウェア記述は、セキュリティコーディングスタンダードに準拠しています。」
[ED-203A / DO-356A]

特定のプログラミング言語のセマンティクスに関連するセキュリティ リスクを開発者が理解していないために、ソフトウェア実装フェーズでソフトウェアバグが混入する可能性があることは広く認識されています。このリスクを緩和する方法の 一つは、セキュリティに重点を置いた適切なコーディングスタンダードを採用することです。コーディングスタンダードの構築は、航空システムの開発に使用されるプログラミング言語の既知の脆弱性を考慮して行う必要があります。採用したコーディングスタンダードが、使用するプログラミング言語に関連するすべての既知のセキュリティ脆弱性をカバーしていると主張する一つの方法は、国際標準化機構 (ISO) 技術レポート 24772 の「言語の選択と使用によるプログラミング言語の脆弱性を回避するためのガイダンス」に記載されている内容に準拠することです。

このレポートでは次のように説明しています。
「プログラミング言語には、仕様が不完全であったり、動作が未定義であったり、実装に依存するため、正しい記述が困難になる要素が含まれています。そのため、記述によっては脆弱性が生じる可能性があり、結果として、ソフトウェアプログラムが作成者の意図とは異って実行されることがあります。場合によっては、この脆弱性がシステムの安全性を損なったり、攻撃者に悪用されてシステムのセキュリティやプライバシーを侵害されたりすることもあります。」

さらに、このレポートでは、脆弱性につながる可能性のあるプログラミング言語構造を特定し、安全なアプローチを提供して、一般的な脆弱性を回避する方法を示します。

4.1.3 数（Quantity）と品質（Quality）

反証（Refutation）テストの考慮事項は、次の 二つの指標に基づいて分類できます。

- 数 （Quantity）：どの程度あれば十分でしょうか？
- 品質（Quality） ：「強力な反証（Refutation）の証拠」とは何を意味するのでしょうか？

確認された反証（Refutation）活動は、必要なテストケースの組み合わせの数が、網羅的なテストを実現不可能にする「ネガティブテスト」の概念 に従っていることが多く、「数（quantity）」は難しい問題です。

機能要件は、システムがどのように動作すべきかを規定し、「ポジティブテスト」によって検証することができます。ネガティブテストは、「仮にどうであったなら（what if？）」に対処することで、要求される動作の逆の動作に焦点を当てます。しかし、要求事項によっては、逆のシナリオは無限にあり得るし、先入観によって、脆弱性の発見につながるシナリオが除外されることもあります。加えて、DO-178C のような安全規格に見られるような従来の適用範囲基準のオブジェクティブは、セキュリティオブジェクティブを念頭に置いて設計されたものではありません。保持された状態や動的なメモリ割り当てを含むシナリオは、カバレッジを測定する際に必ずしも十分に考慮されておらず、セキュリ ティを議論する際に信頼することはできません。このことは、 次のシナリオに示しています：

- ソフトウェアユニットを対象とした一連のテストケースが作成されます。各テスト実行の制御フローを解析することで、カバレッジメトリクスが計算され、100% Modified Condition Decision Coverage（MC/DC）が記録され、バグは検出されません。その後、同じテストケースを用いてアプリケーションを再度テストしますが、今度は、異なるアプリケーショ ンステージに到達することが 可能となるインクリメンタルな実行が行われるように、テスト実行の間にシステム状態を保持します。再び 100%の MC/DC が記録され、今回はバッファオーバーフローの不具合が確認されました。

テストスイートを、考える以上の回数を実行したときにバグが特定できる場合、問題はさらに深刻化します。そのため、「数（quantity）」の考慮事項には、セキュ リティのテストに適したカバレッジ基準を含める必要があります。さらに、どのようなカバレッジ基準であっても、100％を達成しても、システムのセキュア性や 安全性が保証されないことがあります。

品質に関する検討は、個々の反証（Refutation）活動が提供できる安全保証のレベルに焦点を当てます。それぞれの活動に対する反証（Refutation）テストの証拠の質は、測定可能であり、テスト計画の中に明示する必要があります。各活動は、論証を裏付けるためにどのような証拠を示すことができるか、また、その証拠が、その活動でテストしている関連セキュリ ティ対策の特定されたセキュリティアシュアランスレベル（SAL）に、どのように適合するかを検討する必要があります。

最後の検討事項では、各反証（Refutation）テスト活動が反証（Refutation）計画の全体的なオブジェクティブにどのように適合するか、ならびに他のどの活動と関連しているかを検討するよう求められます。一つの反証（Refutation）活動で適切な反証（Refutation）テスト計画を策定できるとは考えられません。むしろ、システムに不適切な動作がないという論証を行い、信頼性を向上するためには、複数の活動を組み合わせる必要があります。この検討事項では、各活動の長所と短所、ならびに活動のオブジェクティブを達成するために使用されるソフトウェアおよびハードウェアツールを明確に理解する必要があります。例えば、静的解析を使用してコーディングの脆弱性（例えば、暗号化されていないパスワード入力フィールドによる SQL インジェクション）を特定し、次にファズテストを使用してソフトウェアバグを特定します。この場合、攻撃手段は同じパスワード入力フィールドであり、攻撃されたバグはサービス拒否攻撃を引き起こす可能性があります。この例では、二つの反証（Refutation）活動を使用して 二つの異なる攻撃可能な脆弱性を特定していますが、一つの活動で両方を特定できるとは限りません。

4.2 活動特有の考慮事項

以下の各考慮事項には、反証（Refutation）活動付属文書内に対応する項目を設け、その考慮事項が特定の活動に関連するかについて説明します。

ガイドラインや考慮事項について、対象となる活動に当てはまらない場合を除き、その考慮事項の結果として、もたらされる行動が、反証（Refutation）テストの計画書にどのように記述するか例を示すことは重要です。

1. 検証と反証（Refutation）の分離

ED-203A / DO-356A では、検証と反証（Refutation）は別々の活動として扱う必要があると規定されており、この推奨事項は直接的な目的ではありませんが、反証（Refutation）活動がこのガイドラインにどのように準拠しているかを理解し、文書化する必要があります。

1.1 検証活動と反証（Refutation）活動が分離されていると論証するために、どのような証拠を示すことができますか？

例えば、要件（セキュリティ関連の要件を含む）の検証に直接焦点を当てていない非機能テストを活動がどのように実行しているかなどです。

1.2 反証（Refutation）活動を実行するプロセスが、攻撃者の視点からテスト実行されていることを保証できることを議論ができますか？

ここでの目的は、脆弱性の特定と防御セキュリティ対策の評価の両方に重点を置いた活動を確実に行うことです。

1.3 この活動では、テストケース内の先入観や偏りにどのように対処しますか？

これは、自動化されたネガティブテストを利用し、悪影響をもたらす可能性のある手動テストを見直すことです。この例では、ファズテストに採用されているミューテーションアルゴリズムからの自動テストケース入力を生成します。さらに、テスト入力作成の自動化は、アプリケーション設計の知識により偏る可能性のある手作業による入力作成を切り離す必要があります。

2. 保証の目標

反証（Refutation）テストはセキュリティアシュアランスの議論を支援するために利用され、特定の活動はセキュリティアシュアランスのさまざまな観点に対して行われることが期待されます。例えば、形式証明はシステム/システムの一部に実行時エラーが存在せず、脆弱性がないことを証明するために使用できます。ファズテストでは、バッファ オーバーフロー攻撃などの一般的な脆弱性をターゲットにするために特定のアルゴリズムが使用される場合があります。

これはさらに、PSecAC 内の反証（Refutation）計画で回答ならびに文書化する必要がある三つの考慮事項に分類されます。

2.1 反証（Refutation）活動のオブジェクティブは何ですか？ どのように確認できますか

2.2 その活動がシステム内で実行する予期しない動作とは何ですか？

2.3 その活動は既知の脆弱性/不具合を対象にしていますか？ それとも、ブルートフォース テストケース インジェクションなどの手法を使用して、潜在的な脆弱性を検証しますか？

3. 制限事項

活動の制限事項を明確に理解し、文書化する必要があります。

3.1 反証（Refutation）活動の制限とは何ですか？

例えば、バッファ オーバーフローを識別するためにファジングテストを使用する場合、テスト アプリケーションのランタイムでは常にバッファ オーバーフローが検出されることを保証できますか？
これは Ada で開発されたアプリケーションの場合は当てはまるかもしれませんが、C で開発されたアプリケーションの場合は必ずしも当てはまりません。

3.2 その活動のオブジェクティブ達成を妨げる可能性のある既知の要因は何ですか？

例えば、実行時エラーが存在しないことを証明するために形式証明を利用する場合、形式証明の静的アナライザーは、すべての検証シナリオが網羅されたことを保証するために必要な検証条件を満たすことができますか？

3.3 反証（Refutation）活動の機能に影響があるハードウェア、またはシステム設計アーキテクチャの考慮事項はありますか（命令セットアーキテクチャ、プログラミング言語、またはコンパイラの最適化）？

例として、ファジングテスト中にすべてのソフトウェアバグを特定するためにランタイム検証を実施する場合がありますが、これはランタイム検証が含まれない最終ビルドを想定していません。この検証シナリオでは、検証のパフォーマンスへの影響を考慮する必要があります。

4. システムの複雑さ

システムの複雑さは、システムの脆弱性を特定する活動について直接的な相関関係があります。また、対象となる防御手段の有効性の評価ついても同様です。特定の反証（Refutation）活動の妥当性と有効性を評価するには、システムの複雑さの影響を理解する必要があります。

この点では、システムの複雑さは特定の活動に関連し、活動ごとの複雑さの評価は活動ごとに異なります。例えば、ペネトレーションテストでは、特定のソフトウェアバージョンに対する既知の不具合を特定するために、コードに埋め込まれたパブリック ライブラリ バージョンの可用性に注意を払う必要があります。ファズテストで、複雑な型の比較ステートメント（comparison statements）を使用するコードを複雑と見なすことがあります。定理証明器を使用する場合は、状態爆発とループ終了を検証する必要があります。

複雑さの評価が完了したら、複雑さがどの時点で活動の有効性に影響を及ぼし始めるかを検討する必要があります。

4.1 特定の反証（Refutation）活動で、「システムの複雑さ」の定義とは何ですか？

4.2 複雑さが増すと、活動のスケーラビリティに影響がありますか？

5. 信頼性の測定

セキュリティ クリチカル アシュアランス反証（Refutation）活動において、システムのセキュリティ アシュアランスを評価するために、その活動によってどのような証拠を示すことができるかが鍵となります。

5.1 不適切な動作が回避されたことを検証するために、反証保証（Refutation assurance)活動で、何を示すことができますか？

例えば、ファズテスト機能では、実行された制御フローの領域と実行されていない領域を示すことができますか？　形式証明が採用されている場合、システムまたはシステムの一部に実行時エラーが存在しないことの明確な証拠を示すことができますか？

5.2 新たに特定された脆弱性を記録するために、どのようなフォーマットを使用しますか？

5.3 その保証活動の結果を再現し、実現するためにどのような情報を組み入れる必要がありますか？

5.4 その活動ではどのようなカバレッジ基準を使用し、既知の制限は何ですか？

カバレッジ基準アルゴリズムの制限の例については、「数と品質」の章で説明します。

6. 範囲(Scope)

この検討項目では、システムのテスト対象領域の範囲がどのように文書化されているか、活動では、システムのどの箇所がテスト可能で、テストできないかに重点を置きます。

6.1 その活動はどのような範囲で実行されますか (システム、サブシステム、アイテムなど)?

既存のプラットフォームのセキュリティ アシュアランスの証拠を確認する場合、反証(Refutation)テストはシステム レベルで実行する必要があります。耐空性セキュリティ プロセスが新規開発の開始時から採用されている場合、サブシステムまたはユニット ベースの反証(Refutation)テストを通じて、システムのテスト対象の範囲を縮小できます。活動がシステム全体をテストしていない場合、その範囲はどのように文書化されていますか?

6.2 当該活動において、セキュリティ対策が、その対策に関連するセキュリティ アシュアランス レベルに応じて十分にテストされていることを示す証拠は何ですか?

セキュリティ対策がセキュリティ資産を保護する役割において効果的であるという保証を示すために反証(Refutation)テストが使用されている場合、反証(Refutation)活動が目的に適合しているという論証を示すことが重要です。例えば、既知の脆弱性一式を使用するペネトレーションテストでは、セキュリティアシュアランスレベル 4 のセキュリティ対策に必要な保証を示すために適切ではありません。

6.3 その活動の対象となるソフトウェア表現レベルは何ですか (ソースコード、実行可能コード、またはその他の中間表現など)?

脆弱性を識別するための静的解析手法は、ソースコードを対象とする傾向があります。ファズテストは実行可能コードに対して実行されます。シンボリック実行などのその他の反証(Refutation)テスト活動は、エミュレートされた環境を通じて実行される中間表現を対象とすることがあります。

6.4 特定のソフトウェア表現レベルで活動を対象にする場合の制限は何ですか? また、これはシステムの動作にどのような影響を与えますか?

例えば、反証(Refutation)テストがバイナリ実行可能イメージではなく、システム ソースコードの中間表現または実際のソースコードに対して実行される場合、バイナリ実行可能ファイルを作成するために必要なコンパイル フェーズ中に追加の脆弱性が作成されないという保証はどの程度示されるでしょうか?

6.5 ハードウェア テストハーネスは最終的な実装を表していますか？ 異なる場合、システムの動作にどのような影響がありますか？

例えば、テスト対象のシステムをホスト環境でエミュレーションしている場合、エミュレータがターゲット ハードウェアを正確に表現していることを証明する必要があります。

6.6 その活動では、テスト対象のソフトウェアを変更する必要があり（例、データ収集のためのインストルメンテーションコードを挿入する）、その変更はシステムの挙動にどのような影響を与えますか？

例えば、ファザー（Fuzzer）が制御フローパスの実行追跡のためにバイナリ コードを挿入する場合、追加の挿入命令によって元のプログラムの動作が誤って変更されないことを保証することが必要です。

6.7 その活動はブート プロセスをテストし、整合性が保護されていることを検証できますか？

システムは完全なエンドツーエンド　セキュリティが要求され、起動プロセスにおける防御的セキュリティ対策の完全性が損なわれないことを保証することも含みます。例えば、システムがドライブ暗号化スキームやファイアウォールを実装している場合、セキュリティ対策をバイパスする動作が想定されます。

6.8 **その活動はシステムのどの部分を検証できないのでしょうか？**

例えば、ブート プロセスの完全性をテストするためにファジングは適していません。一方、保証レベルの低いソフトウェア ライブラリを含むシステムに形式証明を適用しても、保証には限度があります。

7. 脅威のシナリオ

脅威のシナリオが特定された場合は、実装されたセキュリティ対策を対象とした反証（Refutation）活動を実行する必要があります。

7.1 **脆弱性評価に対応するためには、この活動をどのように構成すればよいでしょうか？**

例えば、脆弱性文書に記載されている脆弱性が攻撃されないことを活動で実証できるかどうか。

7.2 特定された脅威シナリオのうち、活動でカバーできないものは何ですか？

脆弱性評価では、各脆弱性の攻撃可能性を評価するためにどのような反証（Refutation）活動が採用されているか明確にする必要があります。

8. 経歴

この考慮事項は、実際の反証（Refutation）テスト活動の保証、特にテストの実行に必要なソフトウェアおよびハードウェア ツールに重点を置いています。反証（Refutation）テストツールは DO330 ツール認定レベル（TQL）に従って実装する必要はありませんが、ツール アシュアランスは反証（Refutation）テスト計画に文書化する必要があります。

以下の考慮事項は、特に TQL 評価（assessment）を示すことができない場合に、反証（Refutation）テストツール アシュアランスの評価を分類します。

8.1 航空安全の耐空性を確立するために以前、使用されたことがありますか？

8.2 脆弱性識別の確立したアプローチですか、それとも試験的なアプローチですか？

8.3 外部の組織（会社）が関係する場合、協力会社はどのような証拠（認証パックなど）を示すことhができますか？

8.4 この活動にはどの程度エンジニアによる介入が必要で、オペレーターが関連ツールに関する経験を持っていることをどのように証明できますか？

9. 他の反証（Refutation）活動との関係

多くの場面において、単一の反証（Refutation）テスト活動だけでは、適切なセキュリティアシュアランスを示すことができません。さらに、反証（Refutation）テスト活動の多くは、他の反証（Refutation）テスト活動で達成できる制限が含まれています。例えば、アプリケーション コードに実行時エラーがないことを証明するために、形式証明を使用した場合、サードパーティのライブラリに対する保証を提示できないことがあります。このような場合、ファズテストを適用することで、システムの安全性をより確実にすることができます。

9.1 この活動を補完する他の活動は何ですか？ また、この活動の既知の制限を補うことができる他の活動は何ですか？

付属文書

ファズテスト

A.1 活動の説明

従来のファズテスト（「ファジング」とも呼ばれる）は、ユーザーが準備するテストケース（開始コーパスとも呼ばれる）のミューテーションを通じてシステム入力の自動生成ができるソフトウェア テスト手法です。

ファジングテストは、検証よりも堅牢性に重点を置いています。このテストの目的は、異常な動作条件（制限のないデータを含むテストケースなど）下で、システムが特定のセキュリティ資産を保護できるかを判断します。テスト実行中にシステムがクラッシュ、またはハングしたかを検出する監視プロセスによって実現されます。ファジングを使用すると、契約（contract）による事前条件と事後条件の使用など、状況によっては、システムが不適切な状態、または未知の状態に移行したことを検出できます。ただし、ファジングでは、実装が機能仕様に準拠していることを保証できません。形式検証を採用して実現する必要があります。

ファズテストという用語は、広範囲の自動テスト機能を含んでいます。特に、ファズテストの方法論は、ブラックボックス、グレーボックス、またはホワイトボックスのいずれかになります。ブラックボックス ファズテストでは、アプリケーションの内部動作（つまり、ソースコード）に関する知識がなくても、ソフトウェアシステムをテストできます。ブラックボックス ファズテストでは、ソフトウェアシステムのバイナリ実行可能ファイルは通常、エミュレートされた環境または仮想マシンを通じて実行されます。グレーボックス ファズテストでは、テスト対象システムのインストルメンテーションが必要なため、ソースコードを使用します。インストルメンテーションは通常、コンパイラ ラッパー経由で挿入され、システムの機能仕様に関する知識は不要です（ソースコードをコンパイルや実装の理解は不要です）。ファズテストは、サブシステム、アイテム、または、ユニット内のバグを特定するためにも使用できます。ホワイトボックス ファズテスト アプローチを採用して、正しいテスト範囲が定義され、特定のサブプログラム シグネチャがテスト インジェクションの対象にできます。ホワイトボックス ファズテストは、テスターが、ファズしたいコード内の領域（セキュリティ資産を保護するセキュリティ対策など）を特定するのにも役立ちます。手動（または、可能な場合は自動）でテストケースを作成して開始コーパスに追加して、必要なカバレッジが達成されているかを確認できます。

多くの場合、攻撃者はブラックボックス ファジング機能しか利用できないと思われますが、バイナリ実行ファイルからコードをリバース エンジニアリングする可能性がある場合には、この機能に依存するべきではありません。さらに、テスト対象のシステムに公開されているソフトウェア ライブラリが組み込まれている場合があり、ライブラリに脆弱性（既知または未知）が含まれている可能性を考慮する必要があります。公開ライブラリが使用されている場合、攻撃者はライブラリの完全な API に対してホワイトボックスおよびグレーボックス ファジングテストを実行するためのアクセス権を持ち、発見された脆弱性を利用することができます。

American Fuzzy Lop（AFL）[9]は、ブラックボックス、グレーボックス、ホワイトボックスのファズテストをサポートするファズテスト ミューテーション エンジンの一つです。

A.2 効果的なファズテストを実現するための考慮事項

ファジングは、手動の堅牢性テストや脆弱性識別メカニズムが拡張できない場合に使用されます。これは、大規模なアプリケーションや複雑なコード セマンティクスを持つアプリケーションで起こります。この場合、プログラムを十分にテストするためのテストケースを手動で作成するのは困難となります。さらに、ファジングは、エラーが発生しやすい手動テスト生成を不要にします。

ファズベースのテストで脆弱性の効果を上げるために、テスト環境で利用可能な異常検出器を最大限に活用する必要があります。例えば、Ada などのプログラミング言語には、複数の制約（メモリ オーバーフロー チェックや開発者が組み込んだコントラクト アサーションなど）のランタイム チェックが含まれ、これらをファズテストハーネスに含める必要があります。C などのランタイム チェックが制限されている言語の場合は、サードパーティの異常検出器（無効なメモリアクセス検出器など）を利用する必要があります。異常検出器を利用しないと、偽陰性が発生し、セキュリティアシュアランスのレベルが不正確になる可能性があります。

A.3 カバレッジとファジングキャンペーンの基準

多くのファズテスト機能は、ファズテスト セッション中に実行されたテスト対象システムのカバレッジ レベルをある程度予測しています。ファズテストのカバレッジは、ツールに依存せず、テストハーネスのコンパイルで追加された計測ポイントの測定から、カバレッジの詳細な解析を行うなど多岐にわたります。カバレッジ アルゴリズムには制限があり、MC/DC カバレッジ などの厳密なアルゴリズムに基づいて最大カバレッジを達成しても、システムに脆弱性がないことが保証されないことは前述しています。ただし、MC/DC のアルゴリズムが DO-178C でレベル A ソフトウェアのテストを確実にするために義務付けられている一方で、カバレッジはソフトウェアテストの有効な指標としてみなされています。

ここでのファズテストによるカバレッジの指針は、DO-178C のカバレッジオブジェクティブの手法を採用する一方で、適切な時間内に 100％の MC/DC カバレッジはファズテ ストでは達成できないことを示しています。したがって、同等の DAL（Development Assurance Level）のオブジェクティブを達成するのと同様に、テスト対象システムのセキュリティアシュアランスレベル（SAL）を考慮しなければなりません。例えば、ファズテストは、安全性が重要な DAL コンポーネントに関連するプロパティを持つセキュリティ資産を対象としている場合、関連する安全上の危険がないセキュリティ対策のファズテストと比較して、より高い保証が必要になります。カバレッジを高め、ファズテストの実行時間を長くすることで、より高い保証を実現できます。

また、多くのファジングテスト機能では、ステートメントカバレッジ以外の検証ができないことがあります。その理由は、少なくとも一般的な「グレーボックス、パスを認識する 」ファジングソリューションでは、システムの異常を特定しないテストケースは、カバレッジ解析のために保持されず、また制御フローの新しい実行パスも発見できないからです。通常、American Fuzzy Lop [9] などのツール上に構築されたファジング ソリューションでは、制御フローを通じて新しいパスを識別できるかは、ヒットしたインストルメンテーションポイント（基本ブロックの周囲に配置）の解析によって決定されます。これは、実質的にアセンブリ言語レベルでのステートメントカバレッジの測定となります。MC/DC を測定するには、変更されたすべてのテストケースで解析を実行する必要があります（どのパスをたどったかに関係なく、簡単ではありません。通常、新しいパスを発見できないテストケースは、テスト実行後に破棄されるためです）。

したがって、ファズテストでは、ステートメントカバレッジを考慮し、開始コーパスの品質の指針としてのみ使用できます。したがって、次の 2 段階の手法を採用することを推奨します。

1. ステートメントのカバレッジレベルを達成するテストケースの開始コーパス、または、反証（Refutation）テストが必要とされるアプリケーションの領域を十分にターゲットとするテストケースの開始コーパスの作成に取り組みます。
2. 開始コーパスを利用してファジングキャンペーンを実行し、キャンペーンの停止基準を定義するための式を導き出します。

実際には、これには「開始基準」と「停止基準」の定義が必要で、両方とも反証（Refutation）テスト計画で明確に定義する必要があります。また、テスト対象システムをターゲットにするために、複数のファジングキャンペーンが必要になります。この場合、それぞれのキャンペーンを検討し、キャンペーンごとに適切な「開始基準」と「停止基準」を定義します。

適切な開始コーパスを作成するために、次の二つの方法を推奨します。

1. 可能な場合は、必要なレベルのカバレッジを達成できる、または、ファジングキャンペーンが反証（Refutation）するシステムのセキュリティをターゲットにする既存のテストケースを採用します。
2. 開始コーパスを手動で作成した後、ファジングツールを使用し、コーパスをより広範囲に及ぶテストケースに変換します。これにより、必要なレベルのカバレッジが達成される、あるいは、ファジングキャンペーンが反証（Refutation）するシステムの特定のセキュリティをターゲットにできます。

上記の方法 1 は、統合テスト、または単体テストがシステムにすでに適用されている（または適用される予定）場合に、最も可能性の高い解決策と考えられます。これは、安全性の保証を論証するためにテスト検証方法を使用して、安全性の保証レベルが高いシステムに当てはまります。ただし、単体または統合レベルのテストケースをファジングテストに適したテストケースに変換するためには、ある程度の時間を要しますが、変換は自動化できます。

開始基準が導き出され、確立され、管轄機関によって承認されると、実際のファジングキャンペーンを実行できます。ただし、システムに脆弱性が含まれるリスクが許容範囲内であると十分に論証するために、キャンペーンの停止基準も明確に定義する必要があります。

この式では、以下の考慮事項はありますが、次の三つのファジングキャンペーンの要件が生じます。

1.開始基準を満たしている。
2.テストハードウェアに十分な能力がある(1 秒あたりのテスト実行回数で測定)。
3.キャンペーンの実行に必要な時間が計算されている。

上記の項目 2 と 3 は密接に関連しており、「必要なテスト実行回数が計算されている」という要件の下にグループ化できます。

上記の測定値を導き出す式を決定するために、考慮事項は以下です。

- テストケースの最大順列(計算可能な場合)
- 入力データの複雑さ
- テスト対象システムの複雑さ
- 達成可能な 1 秒あたりのテスト実行数(ハードウェア能力、プロセッサ数、コア数、テストケースごとのフォーク/実行数などの要素を考慮する必要があります)。
- テストケースシードごとに実行されるミューテーション
- 採用したミューテーションアルゴリズム手法(決定論的、ランダム、構造認識など)
- 必要なコーパスサイクル(コーパス中のすべてのテストケースが完全なミューテーションサイクルを経た回数)
- 対象としているセキュリティ対策のセキュリティアシュアランスレベル

停止基準の計算式の例は、以下の考慮事項のガイドラインセクション 6.2 に示しています。

A.4 計画での考慮事項

反証(Refutation)のために採用されたファジング手法 (以下のセクションでさらに説明) は、PSecAC 内、ならびにファジングキャンペーンごとに適用されます。

- ミューテーションアルゴリズム(Mutation algorithm)の定義
  - 採用された各アルゴリズムの説明
    - サードパーティのファジング ミューテーション エンジンに組み込まれているアルゴリズム(例えば、AFL を使用する場合は、使用したツールのバージョンとミューテーション フェーズ(例: ~deterministic / havoc など)を指定します)

- アルゴリズムのカスタマイズ（つまり、ミューテーションの無害化）
- 開発されたプロジェクト固有のアルゴリズム（例: プロトコル対応のミューテーション）

- 開始コーパス生成ストラテジー
  - 開始コーパスを生成するために使用されたメカニズムの説明
  - カバレッジ解析の実行方法の説明（該当する場合）

- 異常検出メカニズムの強化
  - 脆弱性を検出するために使用されるランタイムチェックの説明
    - プログラミング言語によって強制される実行時チェック
    - 追加のサードパーティ脆弱性検出（例：AddressSanitizer）

- テストケースミューテーションを含むテストケース生成機能
  例:
  - シンボリック実行
  - RedQueen のような「マジックバイト」を見つけるツール[7]

- 停止基準式とその結果としてのキャンペーン必要条件指標

A.5 ガイダンス

1.1 検証活動と反証（Refutation）活動が分離されていることを論証するために、証拠を集めることができますか？

ファズテストは、従来の検証活動とは本質的に異なります。検証テストは、機能要件が満足した証拠を示すために重点を置いています。一方、ファズテストは、システムの正しい機能動作に焦点を当てるのではなく、異常な動作条件下でのシステムの堅牢性を検証するために使用されます。そのため、ファズテストは、ソフトウェアバグ（脆弱性として分類される場合とされない場合があります）を特定するための優れたツールになります。特に、障害の発生経路となる原因が従来の検証テストケースでは通常考慮されない場合に有効です。
さらに、機能要件に関する事前の知識や影響がないテストケースの自動化は、活動が検証とは別であることを明確に示しています。このように独立性を確保する一つの方法は、多くのテスト決定をテストエンジニアからファジングエンジンに委ねることです。ファズテストでは、テストケースを動的に自動生成します。無差別自動化、特に自動化エンジンが入力データ形式またはテスト対象システムのアーキテクチャに関する予備知識を持たない場合、担当者の前提条件がテストに影響を与えるリスクを低減します。

ファジングテストには、テストケースの動的かつ自動生成され、担当者のテストケース作成の負担が緩和されます。ただし、効果的なファジングエンジンは、手動で生成される、あるいは。そうではない開始コーパスを使用します。最初のテストケースには、システム実行を完了させる有効なデータが含まれている必

要があります。開始コーパスが有効であるために、各テストケースで、制御フローグラフ内のさまざまなパスからテストを実行する必要があります。これにより、ファジングツールでは、最大カバレッジというオブジェクティブで効果が得られます。

ファジングセッションの範囲とテスト対象システムによっては、テスト対象システムの基盤となるアーキテクチャを理解していない場合でも開始コーパスを作成できることがあります（例えば、API がファジングされている場合）。ただし、制御フローの点で開始コーパスが広範囲に及ぶために、テスト エンジニアがコードも理解する必要があります。

エンジニアが作成した開始コーパスでは、先入観がデータ値に影響を与えることがあり、システム要件に関する事前の知識が、テストに影響を与える可能性があります。開始コーパスからの制御フローパスに沿った分岐は、標準的なテストミューテーションによってファザーの適用範囲内となります。ただし、開始コーパス内の制御フローパスは安全であり、テストケースの対象とならないことがあります。コードが複雑な場合、すべてのパスが実行されるようにファザーがデータをランダムに変更することが難しい場合があります。

手動で開始コーパスを作成する代わりに、開始コーパスを自動生成できます。そのためには、テストケースのテストデータ型コンポーネントに対して複数の値を自動的に生成する必要があります。各テストデータ型に対して生成された値の順列を、開始コーパスが生成できるように作成します。一部のファジングエンジン、特にグレーボックスファザーは、開始コーパスで最大カバレッジを達成するために、必要なサイズに縮小する機能を持っています。この機能が利用できない場合（つまりブラックボックスファジング）、エンジニアが作成した開始コーパスを使用するのが最良の方法です。

反証（Refutation）テスト計画では、コーパス生成を開始する方法を説明する必要があります。
システム要件を熟知していることによって、開始コーパス生成に不適切な誤りが付加されない方法について説明すべきです。

1.2 反証（Refutation）活動を実行するプロセスが、攻撃者の視点からテスト実行されていることを保証できることを議論ができますか？

グレーボックスまたはホワイトボックスのファズテストはブラックボックスのファズテストよりも大きな成果をもたらしますが、システムレベルのブラックボックスのファズテストを追加実行することで効果が望めます。また、攻撃者がグレーボックス、またはホワイトボックスのファズテスト機能を利用する可能性は低いですが、ブラックボックスのファズテストの方を利用する可能性が高いことを、気に留めておいてください。さらに、特定のシステムに適用できるミューテーションアルゴリズムは、広範囲に及ぶテスト入力コーパスを実現する上で重要な役割を果たすことができます。ただし、攻撃者は標準的な汎用アルゴリズムしか利用できない可能性が高いため、攻撃者の考え方を模擬するために、同じ攻撃方法論を試してみることです。

システム固有のミューテーションアルゴリズムは制御フローの深いところまで到達できますが、汎用ミューテーションアルゴリズムの代替手段ではありません。汎用ミューテーションアルゴリズムは、このコンテキストでは、テストケースの特性を考慮していないと解釈できます。システム固有のミューテーションアルゴリズムは構造を認識し、通常はミューテーションを修正して、システム境界でテストケースが拒否されない様にします。
構造を認識したミューテーションが必要な場合は、汎用ミューテーションアルゴリズムと組み合わせて使用することをお勧めします。攻撃者が構造を認識したミューテーション、あるいは反証（Refutation）テスターと同じ構造を認識したミューテーションアルゴリズムにアクセスできる可能性は低いです。ただし、攻撃者は常に汎用ミューテーションアルゴリズムを利用できます。

1.3 、テストケースで、エンジニアの先入観や偏りに対してどのように対処しますか？

ファズテストではテストケースを自動生成できますが、生成されたテストケースに望ましくない偏りが含まれる可能性が二つあります。

- ミューテーションアルゴリズム
- 開始コーパス生成

ファズテスト手法には、ランダム ビット フリップ パターンを含む複数のミューテーションアルゴリズムを組み込むことをお勧めします。テストケースのデータ構造を認識するミューテーションは目的を達成できますが、ランダムミューテーションを含まないた手法は推奨できません。

開始コーパス作成方針は、テスト対象システムのカバレッジを可能な限り達成できるように作成します。開始コーパスが制御フローグラフの特定の部分に重点を置きすぎると、ファジングツールがテストケースシードを変更してフローを横切って、より広範囲の経路に到達することが困難になります。ただし、ステートメント レベルのカバレッジ解析をファジングセッションと開始コーパス生成に組み込むことで、この問題を緩和できます。

2.1 反証（Refutation）活動のオブジェクティブは何ですか？ どのように確認できますか？

ファズテストの主な目的は、生成されたテストケース入力によって、テスト対象のシステムがクラッシュ、または、ハングアップするかを検証しながら、対象となる制御フローグラフに従うテストケース入力を自動的に生成することです。ただし、ファズテストは網羅性テストではないため、コードに脆弱性がないことを保証することはできませんが、大量のテストケース入力を適用した場合にシステムが堅牢であることを示すことはできます。

反証（Refutation）活動としてのファズテストは、システムにソフトウェアバグがないことを確認できます。ファズテストでは、特定のテストケースによって、システムまたはシステムの一部が誤ってクラッシュしたり、実行状態が不定になったりするソフトウェアバグを特定することで、確証が得られます。この動作の特定は、通常、セグメンテーションエラーの検出、または他のアプリケーションレイヤーやオペレーティングシステムレイヤーで検出されたエラーの監視によって行われます。

さらに、一部のファズテストメカニズムでは、システムが指定されたタイムアウト時間内に実行できなかったことを検出する場合もあります。これにより、特定のテストケースデータ入力によって、システムが長時間の処理状態になったり、永久にハングしたり、ブロックされることがなく、サービス拒否（DoS）攻撃が回避されます。

2.2 その活動がシステム内で実行する予期しない動作とは何ですか？

ファズテストでは、システムが堅牢であり、異常な動作状況下でも機能仕様の範囲内で動作を続けることを実証するシナリオを作成します。

特に、ファズテストでは、正しく制御されている、そうではない場合もあり、テスト対象のシステムによって有効と見なされる場合とそうではないこともあるテストケースデータが生成されます。その後、ファズテストメカニズムは、テストケースの処理中にテスト対象のシステムを観察し、クラッシュやプロセス タイムアウトなどの不適切な動作を検証します。

2.3 その活動は既知の脆弱性/不具合を対象にしていますか？ それとも、ブルートフォース テストケース インジェクションなどの手法を使用して、潜在的な脆弱性を検証しますか？

ファズテストは、テスト対象のシステムが誤って実行を停止したり、予期しない実行状態になったり、実行サイクルをオーバーランする原因となるデータ入力を検出します。この予期しない動作状態はソフトウェアバグによって発生します。通常、一般的なソフトウェアバグはファズテストの対象ではないため、予期しない動作の原因となったソフトウェアバグの特定は困難です。しかし、バッファオーバーフローや範囲チェック例外のエラーが発生することを意図した、ミューテーションで利用できる特定の手法があります。

多くの場合、ファズテストは、システムのクラッシュやハングを引き起こすソフトウェアバグを対象としています。

3.1 反証（Refutation）活動の制限とは何ですか？

ファズテストセッションの作成を可能な限り自動化するソフトウェアツールが存在する一方で、このテクノロジを採用しにくくするいくつかのシステム上の特徴があります。
一般的にファジングは、以下の伝統的なモデルに準拠したシステムに適しています：

1. 入力を受け取る
2. 入力を処理する
3. 停止

このモデルに適さないシステムは変更が必要です。例えば、バッファ キューからのメッセージを処理するループを持つアプリケーションは、メッセージが受信され処理された後に実行が完了するように変更する必要があります。ただし、これは、標準的な単体レベル、ならびに統合レベルのテストの適用範囲と変わりません。

さらに、ミューテーションアルゴリズムに情報を提供するために、テスト対象システムのインスツルメンテーションに依存するファズベースのテクノロジもあれば、新しいテストケースプロセスを効率的に生成するために、POSIX フォークサーバなどの複雑なテストドライバを必要とするものもあります。このような「グレーボックス」スタイルのファジングアプローチが採用されると、テスト対象システムは、ターゲット実行可能コードの真の表現ではなくなり、ファジングセッションの必要な動作パラメータ内でシステムが実行できることを保証するために、異なるコンパイラが必要になります。

ファズテストに関するもう 一つの考慮事項は、いつ停止するか確認することです。ファズテストのミューテーションアルゴリズムは、特にテストケースの入力構造が単純であると考えられる場合や、カバレッジ解析ですべての入力が生成されたことが確認された場合に、枯渇する可能性があります。ただしテストケースの入力データが複雑であるため、テストケースのミューテーションの組み合わせが膨大になることがあります。この手法では、テストエンジニアが、テストを停止できるタイミングを規定するための明確な基準が必要です。複雑な停止基準ルールを定義する自動化機能を提供するファズテストソリューションは、この問題に対する部分的な回答を示します。ファズテストを含む反証（Refutation）テスト計画の承認を得るために、ファズテストツールで適用できる停止基準アルゴリズムを文書化する必要があります。停止基準アルゴリズムの例は、セクション 6.2 で定義されています。

ファズテストセッションの終了後、達成されたカバレッジを評価し、該当する場合は、手動で作成した追加のテストケースを開始コーパスに追加して、対象のコードの未変更領域で実行します。その後、ファズテストセッションを再開します。このプロセスは、開始コーパスに必要なカバレッジが達成されるまで繰り

返します。開始コーパスの基準を満たしていることが確認できたら、ファズキャンペーンを開始できます。キャンペーンは、停止基準を満足すると終了します。

3.2 その活動のオブジェクティブ達成を妨げる可能性のある既知の要因は何ですか？

システムの複雑さやデータ入力の複雑さ（検討事項 4 で定義）が増加するにしたがって、ファジングセッションで必要なレベルのコードカバレッジを達成できる可能性は低くなります。複雑さ、またはテストケースカバレッジのいずれかを測定するメカニズムが必要になります。ファジングで必要なカバレッジを達成できないシステムは、分割してファジングテストを実行する必要性があります。

3.3 反証（Refutation）活動の機能に影響があるハードウェア、またはシステム設計アーキテクチャの考慮事項はありますか（命令セットアーキテクチャ、プログラミング言語、またはコンパイラの最適化）？

一般的に、ファジングテストに適しているプログラミング言語があります。特に、制約チェックを含めるように構成できるランタイムを持つ言語は、ファジングテストの目的に最適です。契約（contract）による設計などの概念をサポートする言語も適しています。バッファ オーバーフローなどの脆弱性によってシステムが未知の状態に陥る可能性の高い言語は、あまり適していません。この例として、バッファ オーバーフローを識別可能なのは Ada ランタイムで、C ランタイムはそうではありません。

一部のファズテスト アーキテクチャでは、コード インストルメンテーションが必要としますが、多くの場合、プロセッサ エミュレータを使用したテスト実行もサポートされています。
ファズテストでは、テスト対象のシステムと並行して構築、または実行する必要がある複雑なテストハーネスラッパーが必要になります。そのため、航空宇宙の電子機器で一般的に使用される組込システムのファズテストは、難易度が高くなります。この場合の解決策は、多くの場合、ターゲットハードウェアのエミュレーション環境を用意して、ファズテストを移植することです。ただし、セクション 3.1 で説明したように、組込システムでは、ファズテストを実行するために変更が必要になる場合があります。

さらに、特にファズテストハーネスがターゲット コンパイラから利用できないライブラリ コンポーネントを必要とする場合、必要なコンポーネントをサポートできるホスト環境でアプリケーションをクロスコンパイルするという代替策もあります。例としては、AFL が POSIX *fork()*および *exec()*関数の実装に依存していることが挙げられます。

4.1 特定の反証（Refutation）活動で、「システムの複雑さ」の定義とは何ですか？

複雑さは、ファジングセッションで、すべての脆弱性が見つかる可能性を決定する重要な要素です。ファジングテストの範囲では、複雑さという用語は 二つのカテゴリに分けられます。

1. テスト対象システムの複雑さ

コードのデシジョンの数と階層の深さ、つまりデシジョンポイントの数とデシジョンを構成する条件の組合せの数を考慮して測定されます。さらに、従来は複雑とは見なされていなかった条件、つまり広範囲の境界を持つデータの等価性テスト（浮動小数点の等価性など）も、複雑さと見なされ、ブラックボックステストで完全なカバレッジを達成するのが困難になる条件も含まれます。

ファズテストの複雑さに影響を与える、その他の要因には、システムのアーキテクチャ内の同時実行性のレベルや外部インターフェイスの数などがあります。

2. テストケースの複雑さ

テストケースデータの構造を考慮し、個々のスカラー リーフ ノードの定義された境界を含めて測定されます。さらに、データが準拠しているシステムのプロトコルや、データのコンポーネントが、疎結合か密結合かを考慮する必要があります。

テストケースの複雑さに影響を与えるもう一つの要因は、ファザーのミューテーションエンジンに実装されているテストケース生成手法です。構造を意識したミューテーションエンジンを使用するファジングセッションでは、インターフェイスによって捨てられる無効なデータを生成するサイクルの無駄が少なくなります。
ただし、カスタムミューテータは本質的に人に依存するため、常にそれを排他的に使用しないようにすることが推奨されます； 純粋な構造を意識したミューテーションエンジンでは、無効な（制約のない）データを含むテスト・ ケースを通してのみ発見できる脆弱性を特定できない場合があります。

4.2 複雑さが増すと、活動のスケーラビリティに影響がありますか？

テスト対象システムの複雑さが増すと、ファジングツールが目的のレベルのカバレッジに到達する可能性が低くなります。カバレッジ目標に到達するために必要なテストケース生成数が非常に多くなる（数百万以上）可能性があるため、ファジングテストのハードウェアの考慮事項も重要な要素となります。複雑度の高いシステムでファジングセッションを処理するには、マルチコア サーバーが必要になることがあります。

スケーラビリティの決定は、システムの計算された複雑さと、ファジングセッションを実行するハードウェアの要素となります。1 秒あたりのテストケース生成数は、考えられるテスト入力の推定合計の順列(計算可能な場合）を考慮する必要があります。

ファジングツールの多くは実行中に動的なフィードバックを出力しますが、グレーボックス スタイルのファジングが採用されている場合、テストエンジニアは、ファジングツールが制御フローグラフ内で最後に新しい実行経路を見つけてからの経過時間の長さに注意する必要があります。最後の実行経路の時間をテストケース生成速度（ヘルツ）と組み合わせて、現在のアプローチがテスト対象のシステムに対して拡張可能かを判断できます。

セッションに拡張性がない場合は、次の 二つのオプションが利用できます。

1. 複雑さを緩和する必要がある
2. 1 秒あたりのテストケース生成数を増やす

5.1 予期しない動作が回避されたことを検証するために、反証保証(Refutation assurance)活動で、何を示すことができますか？

ファズテスト アプローチでは、通常、予期しないシステム動作を実現できなかったテストケースは破棄されます。これは、生成されたテストケースのすべてをファイルに保存することが現実的ではないため、意図的に行われています。さらに、テストケースがミューテーションアルゴリズムによって動的に作成される場合、このアプローチでは、テストの実行速度に悪影響を与える不要なオーバーヘッドが発生します。したがって、システムの信頼性を確認するために実行可能で、再現可能な回帰テストを作成することは、現実的でないことがあります。

その代わり、ファズテストでは、システムを予期しない状態に移行させるテストケースや、制御フローグラフを通じて新しい実行経路を見つけるテストケースを登録する場合があります。後者はコードカバレッジメトリックスを計算するために使用し、前者は事前に特定された脆弱性が修正されたことを示すために使用できます。カバレッジによってテストの抜けが特定された場合、追加のテストケース シードを作成して開始コーパスに追加し、ファズテストを再度実行できます。適正なレベルのカバレッジが達成されるまでこれを繰り返す必要があります。

5.2 新たに特定された脆弱性を記録するために、どのようなフォーマットを使用しますか？

各ファズテストでは、テストを再実行できるように、開始コーパスと関連するテストハーネスを記録する必要があります。さらに、制御フローグラフを通じて新しい実行経路が見つかったすべてのテストケースを記録するグレーボックス ファザーは、これらのテストケースをファズテストに関連付けられた開始コーパスに保存する必要もあります。

したがって、ファズテストのミューテーションアルゴリズムでは、乱数ジェネレーターを使用して、コーパス内のキューにあるテストケースをテスト実行用の新しいテストケースに変換 (mutate) するのが一般的です。そのため、ファズテストセッションは実際には再現性がないことがよくあります。場合によっては、疑似乱数ジェネレーターを固定シードで使用するように構成して、テストをより決定論的にできます。

ただし、固定シードを使用して決定論的なファズテストを生成することは、推奨される手法ではありません。これは、ランダムシード値を使用すると、興味深いテストケース入力値が生成され、網羅性が高いカバレッジが実現され、より多くの脆弱性が検出される可能性があります。

ファズテストでは、アシュアランス エビデンス ジェネレーションの生成に二つの重要な要素があります。

1. コーパス内でテストケース入力を実行したときに達成されたカバレッジを解析する機能。
2. ファジングセッションの継続が許容できないディメンション リターンのポイントに達したことを示すために使用される「停止基準ルール」の定義。

セクション 3.1 では、停止基準の概念について説明しています。
さらに、テスト対象システムで予期しない動作を発見したテストケースは保存し、後で関連するソフトウェアのバグが修正されたことを確認するために使用する必要があります。

5.3 その保証活動の結果を再現し、実現するためにどのような情報を組み入れる必要がありますか？

システムで予期しない動作を引き起こすテストケースファイルを保存します。このテストケースは脆弱性を再現するために使用できます。さらに、ファズテストコーパス、ファズテストハーネス、および実行環境設定を保持する必要がありますが、これによって動的に生成されたテストケース(セクション 5.2 で説明) と同じものが生成されるという保証はありません。

5.4 その活動ではどのようなカバレッジ基準を使用し、既知の制限は何ですか？

ファズテストは、特定のコードカバレッジ アルゴリズムに縛られることはありません。さらに、ファザーがすべての生成されたテストケースまたは制御フロー グラフを通じて新しい実行経路が見つかったテストケースが保存されると仮定して、ファジングツールとは独立して、保存されたテストケースをテスト対象システムで実行できます。そのため、複数のカバレッジ メカニズムを利用して、システムを通じて達成されたカバレッジが記録できます (ステートメントカバレッジまたは変更された MC/DC カバレッジ)。

したがって、カバレッジ アルゴリズムの制限は、選択された特定のアルゴリズムによって異なりますが、アルゴリズムがバッファ オーバーフローとなるインクリメントなどの原因をどのように処理するかについて考慮する必要があります。

さらに、多くのファジングツールでは、ステートメントカバレッジ以外のものを記録することはあまり意味がありません。これは、開始コーパスの式を導き出すためにのみ使用することをお勧めします。つまり、コーパスが 100%のステートメントカバレッジを達成できることを確認する必要があります。

6.1 その活動はどのような範囲で実行されますか (システム、ユニット、その他)？

その他のソフトウェア テストと同様に、ファジングテストは、関数スタブなどの標準的なソフトウェア境界強化手法(software boundary tightening techniques)を使用して範囲を限定します。効果的な脆弱性テストを実行するために、ファジングセッションの範囲を限定する必要があります。システム レベルのファジングテストが実行不可能な理由は複数あり、次に示します。

- 複数のシステムインターフェースが複数の外部ソースから同時に駆動される並列実行アプリケーション
- 割り込みを処理するために「メインループ」に入ることが多い組込アプリケーション。
  正常に終了できなかったものは、ファザーによって「ハングしたプロセス」と判断
- クローズドソースのサードパーティライブラリを利用するアプリケーション
- テスト対象の入力データのプロトコル制約チェックを含むアプリケーション（例えば、ファズテストミューテーションエンジンがデータのチェックサムを修正できない場合に必要）

実行可能なファジングテストはシステム レベルで実行する必要がありますが、テスト対象のシステムが複雑になると、効果の減少につながります。したがって、ファジングキャンペーンに関係する複雑さのレベルを理解することは、最大限の効果を達成するために重要で、最終的には、テストのセットアップ、スコープ、実行、監視、および停止をどのように行うのが最適かを決定することが重要です。

ファズテストは、最低限、テスト対象のソフトウェアシステムへの特定された攻撃手法（ソケット接続など）に対して実行する必要があります。理想的には、そのターゲット環境に類似したファズテストハーネスを使用します。

6.2 当該活動において、セキュリティ対策が、その対策に関連するセキュリティアシュアランスレベルに応じて十分にテストされていることを示す証拠は何ですか？

ファジングは、ソフトウェアバグの検出に重点を置いています。これには、異常なデータ入力によって発生する「コーナーケース」の設計エラーや実装エラーが含まれます。これらの入力は、標準的な検証テスト手法では関連性があるとは見なされない可能性があります。

ファズテストでは、テスト対象システムの攻撃対象領域をファズテストのエントリポイントと見なす必要があり、関連するファズテストハーネスは、実稼働するシステムが外部データを受信するために使用されるメカニズムを代表している必要があります。実際のシステム攻撃対象領域からファズテストの挿入ポイントへの経路が常に存在すると想定されます。この意味で、テスト対象システムに挿入される各テストケースは、潜在的な攻撃ベクトルと見なすことができます。アプリケーションが起動して、望ましくない状態、不明な状態、デッド状態、またはブロックされた状態に遷移できる場合、潜在的なセキュリティ侵害またはサービス拒否攻撃が発生する可能性があります。ファズテストでは、関連する攻撃ベクトルを介して攻撃対象領域全体に展開された不具合を介して、これらの攻撃を引き起こす可能性のある脆弱性を検出できます。この意味で、既知の脆弱性に関連するセキュリティ資産を保護するために実装されているセキュリティ対策もテスト対象となります。

特定されたセキュリティ資産の攻撃ベクトルに合致した十分な範囲を活動がカバーしていることを示すために、次の証拠を収集する必要があります。

- 脆弱性が発見された、すべてのテストケース
- 発見された脆弱性が修正された証拠
- テストの範囲と特定された攻撃対象領域（ファズテストの注入ポイント）の説明
- 採用されたミューテーションの説明（潜在的な攻撃手法）
- テストによって達成された範囲
- 対象となるセキュリティ対策が適用された具体的なシナリオまたはシナリオの範囲

さらに、「カバレッジとキャンペーンの基準」セクションで説明があるように、ファジングキャンペーンの開始基準と停止基準では、キャンペーンが対象とする攻撃緩和セキュリティ対策に関連するセキュリティアシュアランスレベルを考慮する必要があります。

これを実現する方法の一例としては、導出された停止基準式内の各変数に、対象となるセキュリティ対策に関連付けられた最高の SAL 番号を乗算します。
このアプローチを使用すると、数式は次のようになります。

$$\sum = \left(\frac{(\alpha \times \beta) + (\gamma \times \beta)}{\frac{\rho}{\beta}}\right) \times \sigma \qquad (1)$$

- α = 制御フローの循環的複雑度（Cyclomatic Complexity）
- β = 対象となるセキュリティ対策に関連付けられた最高の SAL 番号
- γ = 入力データ構造の複雑さ
- ρ = 1 秒あたりに達成可能なテスト実行回数
- σ = 検証結果の評価に使用する任意の数値（例、86400 = 1 日の秒数）
- $\sum$ = このキャンペーンに必要なファジング時間

この式を適用する方法の例を表に示します。ここでは、セキュリティアシュアランスレベルを上げると、ファジングキャンペーンに必要な時間が長くなることがわかります。この例では、開始基準として 100% のステートメントカバレッジを想定していることに注意してください。

$\sum$	β	α	γ	ρ	σ
104 時間	1	4	2	5000	86400
415 時間	2	4	2	5000	86400
933 時間	3	4	2	5000	86400
1659 時間	4	4	2	5000	86400

表 1: 複雑さ、SAL、パフォーマンスを考慮したファジングに必要な時間の例

この例は、考慮事項 6.2 に対する 一つの解決策を示していますが、監督機関によって正しいアプローチとして検証または合意されたものではありません。この例では、制御フローの複雑さと入力データの複雑さを測定する方法についてのガイダンスは提示されていません。プロジェクトに固有の停止基準要件を計算するための独自の式を導き出すのは、利用者の責任となります。

6.3 その活動はどのソフトウェア表現レベルを対象としていますか (ソースコード、実行可能コード、またはその他の中間表現)?

ファズテストでは、テスト対象のシステムを実行する必要があり、ソースコードをターゲット実行ファイルにコンパイルする必要があります。ターゲットコンパイラとは異なるコンパイラが必要になる場合、不要な場合もあります。また、ソースコードレベル、またはより低いレベルで追加のインストルメンテーションが必要になることもあります。
ファズテストは、最終的には実行可能コードに対して実行されます。実行可能コードは、リリースされた実行可能コードと同じである場合もあれば、ファズテスト固有なコードの場合もあります。

6.4 特定のソフトウェア表現レベルで活動を目的にする場合の制限は何ですか?また、これはシステムの動作にどのような影響を与えますか?

ファズテストをホスト環境でのみ実行できる場合は、ホストプロセッサの動作をターゲットプロセッサと比較する必要があります。少なくとも、次の潜在的な違いを考慮する必要があります。

- コンピュータメモリ内のバイトの順序またはシーケンス
- 組込計測機器のパフォーマンスへの影響
- スタブからの戻り値(ファズテストの範囲を縮小するために必要)は、実際のシステムの動作を反映しているか

検証テストについても同様の考慮事項があり、同じプロジェクトのガイダンスに従う必要があります。

6.5 ハードウェア テストハーネスは最終的な実装を表していますか？ 異なる場合、システムの動作にどのような影響がありますか？

ケースバイケースで評価する必要があり、一般的な American Fuzzy Lop (AFL) アーキテクチャ上に構築されたファジングエンジンは、POSIX ベースのオペレーティングシステムでの実行に限定される可能性があります。ファジングセッションがシステム ハードウェアアーキテクチャのホストバージョンでのみ実行できる場合、ターゲット コンパイラの実行可能コード生成によって導入された脆弱性、またはターゲット ハードウェア内の脆弱性が存在する可能性があり、これらはホスト ベースのファジングテストでは検出されません。

6.6 その活動では、テスト対象のソフトウェアを変更する必要があり (例、データ収集のためのインストルメンテーションコードを挿入する)、その変更はシステムの挙動にどのような影響を与えますか？

これもケースバイケースで評価する必要がありますが、ファジングテスト手法では、テスト対象システムの経路を認識するためにフィードバックイインストルメンテーションが必要になることがあります。さらに、ファザーの中には、テストケースの実行時間を短縮するために、テスト対象システムのコードにテストケースの実行構文を組み込むことがあります。

このシナリオでは、実際のアプリケーションをターゲットハードウェア上でファジングテストすることは現実的ではない可能性があり、特に組込フライトソフトウェアに当てはまります。この場合、クロス コンパイルとインストルメンテーション、またはターゲット環境のエミュレーションのいずれかで、ソフトウェアをネイティブ プラットフォームに移植する必要があります。ファジングテストは、テスト対象のソフトウェアが可能な限り高速に完了するまで実行されることも前提としています。システム レベルでファジングテストを行う場合、テスト エンジニアは、コードが確実に実行されて終了するために、コードを手作業で変更する必要があります。例えば、メインループで終了しない組込システムは、ファジングテストには適していません。

ターゲット ベースのファズテストが可能な場合はそれを使用し、ターゲット ベースのファズテストが選択できない場合は、エミュレートまたはホスト ベースのファズテスト アプローチを採用することをお勧めします。

採用した手法がシステムの測定可能なパフォーマンスにどのように影響するかについての説明を、計画で文書化する必要があります。パフォーマンスの低下がタイムクリティカルな動作に影響し、レースコンディションにより、ターゲットリリースに影響しないかを考慮する必要があります。

6.7 その活動はブート プロセスをテストし、整合性が保護されていることを検証できますか？

ファズテストでは実現できません。

6.8 その活動はシステムのどの部分を検証できないのでしょうか？

ファズテストでは、ブートシーケンス中に公開されるセキュリティ攻撃手法（BIOS 攻撃 / ブート ローダー攻撃）を識別できません。
ファズテストでは、ハードウェア攻撃（グリッチ、フォールト インジェクション、サイドチャネルなど）によって露呈される脆弱性を特定することはできません。これには、電力変動による攻撃手法が含まれます。

システムのソフトウェアランタイムにおいて、ファズテストで検証できない場合、短時間で特定できません。ただし、テスト対象のシステムが複雑になるほど、潜在的な攻撃ベクトルをすべて特定することが困難になります。

7.1 脆弱性評価に対応するためには、この活動をどのように構成すればよいでしょうか？

ファジングテストはセキュリティ検証活動であり、主な目的はソフトウェアの脆弱性を発見することです。ファジングセッション中に特定されたソフトウェアバグが、脆弱性評価で特定されていない場合は、評価に対する課題となります。

7.2 特定された脅威シナリオのうち、この活動でカバーできないものは何ですか？

ファズテストは、スタンドアロンソフトウェア アプリケーション内の脆弱性を特定できる可能性がありますが、複数のシステムの相互作用を伴う脅威シナリオは簡単には実行できません。二重または三重の障害条件を伴う複雑なシナリオは、他のソフトウェア検証手法を使用してテストするのが最適です。

電力変動などの方法によるハードウェア攻撃を伴う脅威シナリオには、他のペネトレーションテスト手法の方が適しています。

8.1 この活動は以前に現場で使用されたことがありますか？

執筆時点では、ファズテストは航空宇宙および防衛産業における反証（Refutation）活動として一般的に使用されて実績はありません。しかし、自動車やインターネットベースのクライアント/サーバーアプリケーションなど、他の業界ではより広く利用されています。これは、航空耐空性セキュリティの導入により変更になると考えられます。連邦航空局（FAA）や欧州連合航空安全局（EASA）などの機関がサイバーセキュリティ関連オブジェクトの Acceptable Means of Compliance として Airworthiness Security Process を導入することで、この状況は変わると思われます。

8.2 これは脆弱性識別の成熟したアプローチですか、それとも実験的なアプローチですか？

ファズテストは、すべての航空宇宙および防衛プロジェクトで試行およびテストされているわけではありませんが、人気が高まっています。この手法は、他の分野（自動車、モノのインターネットなど）でより一般的に使用されています。

8.3 外部のパートナーと協業する場合、サブコントラクターはどのようなエビデンス（認証パックなど）を用意できますか？

ケースバイケースで評価されます。

8.4 この活動にはどの程度エンジニアによる介入が必要で、オペレーターが関連ツールに関する経験を持っていることをどのように証明できますか？

エンジニアによる介入は、採用するファジング手法と、使用するファジングテストツールによって異なります。自動化のレベルは、テスト対象のシステムの複雑さとファジングツール テクノロジの機能によって異なります。ソリューションによっては、ほぼ完全に自律的なソリューションを提供するものもありますが、複雑なテストハーネス、カスタムミューテーションアルゴリズム、開始コーパスの構築を必要とするものもあります。

ファズテストハーネス(標準の検証テストハーネスと比較すると)の開発は複雑で、選択したファズテストツールでは、自動化コード生成、テスト実行、および結果照合の複数のレイヤーをサポートすることが推奨されます。

スタッフの適性はケースバイケースで評価する必要がありますが、ファズテスト ソリューションが自動化されると、スタッフがツールを操作するために必要な経験は少なくなる傾向があります。

9.1 この活動を補完する他の活動は何ですか？ また、この活動の既知の制限を補うことができる他の活動は何ですか？

複雑なシステムでは、ファズテストでは必要なレベルのコードカバレッジに到達するために十分なテストケースが生成されない可能性があります。

このような場合、他のカバレッジ測定可能な反証（Refutation）活動（静的解析や形式証明など）を使用してアプローチを補完する必要があります。

ファズテストはペネトレーションテストの代替ではありません。二つの活動は重複する場合もありますが、異なる潜在的な攻撃手法に重点を置く傾向があります。特にペネトレーションテストは、ブート読み込みシーケンス内の脆弱性や特定のオペレーティングシステム内の既知の脆弱性を攻撃する場合に使用で

きます。さらに、ペネトレーションテストと静的解析は既知の脆弱性の特定に重点を置きますが、ファズテストはあらゆる脆弱性（これまで知られていなかった脆弱性を含む）を発見できる可能性があります。

ファズテストでは、ランタイムエラーがないことが形式証明されているシステムに対しては、ほとんど効果がない可能性があります。したがって、形式検証されたシステムの高い整合性に対してファズテストを行うことで得られるメリットについて検討する必要があります。例えば、システムがプログラミング言語 SPARK で実装され、SPARK 検査器によってランタイムエラーがないことが形式証明されている場合、ファズテストによってそれ以上のセキュリティアシュアランスを得る可能性は低くなります。

さらに、ペネトレーションテストなどの他の反証（Refutation）テスト活動を使用して、特定されたすべてのハードウェア攻撃手法を網羅する必要があります。

参考文献

[1] RTCA. “”Software Tool Qualification Considerations Corrigendum 1””. In: DO330 (Dec. 2011), pp. 1-5.
[2] Safety-Critical Working Group RTCA SC-167 of RTCA and WG-12 of EUROCAE”. “”Software Considerations in Airborne Systems and Equipment Certification””. In: RTCA DO-178C (Dec. 2011), pp. 1-1.
[3] EUROCAE. “”Airworthiness Security Process Specification””. In: ED-202A (June 2014). for the European Union Aviation Safety Agency (EASA), pp. 1-1.
[4] RTCA. “”Airworthiness Security Methods and Considerations””. In: DO-356A (Sept. 2014). for the U.S. Federal Aviation Administration FAA, pp. 1-1.
[5] RTCA. “”Airworthiness Security Process Specification””. In: DO-326A (Dec. 2014). for the U.S. Federal Aviation Administration FAA, pp. 1-1.
[6] EUROCAE. “”Airworthiness Security Methods and Considerations””. In: ED203A (June 2018). for the European Union Aviation Safety Agency (EASA), pp. 1-1.
[7] Cornelius Aschermann et al. “REDQUEEN: Fuzzing with Input-to-State Correspondence”. In: Jan. 2019. DOI: 10.14722/ndss.2019.23371.
[8] EUROCAE. “”Software Tool Qualification Considerations Corrigendum 1””. In: ED-215 (Feb. 2021). for the European Union Aviation Safety Agency (EASA), pp. 1-5.
[9] Lcamtuf. American Fuzzy Lop (2.52b). (accessed 23 Sep. 2019). URL: http://lcamtuf.coredump.cx/afl.

AdaCore

IT Access®
アイティアクセス株式会社

本社　〒222-0033 神奈川県横浜市港北区新横浜 3-17-6
TEL:045-474-9095 FAX:045−474−8823

URL: https://www.itaccess.co.jp

202410