HTTP/3・QUICに対するキャッシュポイズニング攻撃についての手法と攻撃シナリオの考察

新開 光平[1,a)]　山崎 進[2,b)]

概要：QUIC は Google が開発した高速通信プロトコルであり，QUIC を用いた HTTP/3 は現在ウェブ全体の約 35 %で利用されている．その高速化と最適化の設計はキャッシュの検証や整合性確認を困難にし，Web キャッシュに偽のデータを挿入するキャッシュポイズニング攻撃のリスクを高める可能性がある．しかしながら HTTP/3・QUIC に特化したウェブキャッシュポイズニング攻撃に関する研究は少なく，具体的な対策も十分に整備されていないのが現状である．本研究ではこのギャップを埋めるため，1. HTTP/3・QUIC 通信におけるキャッシュポイズニング攻撃の手法の分析，2. 具体的な攻撃経路とメカニズムの考察を行った．方法として攻撃ベクトルの特定を目的とした文献調査を実施し，特に QUIC による高速通信のための機能である 0-RTT(Zero Round-Trip Time) を利用した攻撃に焦点を当てて攻撃シナリオを考察した．本研究によって HTTP/3・QUIC 環境における新たなセキュリティリスクを浮き彫りにし，今後の防御策構築に貢献することを目指す．

キーワード：HTTP/3, QUIC, Cache Poisoning, 0-RTT

Analysis of Cache Poisoning Attacks in HTTP/3 and QUIC and Consideration of Attack Scenarios

Kohei Shinkai[1,a)]　Susumu Yamazaki[2,b)]

Abstract: QUIC is a high-speed communication protocol developed by Google, and HTTP/3 using QUIC is currently used in about 35 % of the entire Web. Its high-speed and optimized design makes cache verification and integrity checks difficult, and may increase the risk of cache poisoning attacks that insert fake data into the web cache. However, there is little research on web cache poisoning attacks specific to HTTP/3 and QUIC, and concrete countermeasures are not well developed. In order to fill this gap, we 1) analyzed methods of cache poisoning attacks in HTTP/3 and QUIC communication, and 2) examined specific attack vectors and mechanisms. As a method, we conducted a literature survey to identify attack vectors and discussed attack scenarios with a particular focus on attacks using 0-RTT(zero round-trip time), a feature of QUIC for high-speed communication. This study aims to highlight new security risks in the HTTP/3 and QUIC environment and to contribute to the development of future defenses.

Keywords: HTTP/3, QUIC, Cache Poisoning, 0-RTT

1. はじめに

HTTP(Hypertext Transfer Protocol) はインターネット通信において広く使われている．HTTP にはバージョンがあり，現在は HTTP/1.1 と HTTP/2 が広く利用されているが，近年登場した HTTP/3 の普及が進んでいる．W3Tech

[1] 北九州市立大学　国際環境工学研究科
, The University of Kitakyushu
[2] 北九州市立大学　国際環境工学部
, The University of Kitakyushu
[a)] f4mcb007@eng.kitakyu-u.ac.jp
[b)] zacky@kitakyu-u.ac.jp

の調査 [1] によると HTTP/3 の普及率は 2025 年 6 月時点で 35.2 %となっている．HTTP/3 では下位層プロトコルで TCP の代わりに QUIC と呼ばれる UDP ベースのプロトコルを使用している．QUIC は Google が開発した高速通信プロトコルであり，2021 年に RFC9000[2] として標準化された．QUIC の様々な機能によって高速通信が実現できる一方で，いくつかのセキュリティ上の課題が存在する．その中でも QUIC に対するキャッシュポイズニング攻撃についてはあまり研究されていない．

そこで本稿では，セキュリティ課題の一つであるキャッシュポイズニング攻撃に焦点をあてて研究を行った．具体的には　1.HTTP/3・QUIC 通信に対するキャッシュポイズニング攻撃の手法の分析，2. 具体的な攻撃経路とメカニズムを解明し脆弱性を明らかにすること，この 2 点を行った．研究方法として攻撃ベクトルの特定を目的とした文献調査を行い，0-RTT データを利用した攻撃に焦点を当てた．本稿の構成として，2 章にて HTTP/3・QUIC 通信，3 章にてキャッシュポイズニング攻撃について解説する．4 章では先行研究から HTTP/3・QUIC 通信におけるキャッシュポイズニングの可能性を分析し，5 章にて攻撃シナリオの考察を行い，6 章でまとめと将来課題を述べる．

2. HTTP/3・QUIC 通信の概要

2.1 QUIC について

QUIC は HTTP/2 までのバージョンで課題となっていた HoL(Head of Line) ブロッキング問題を解決するために Google によって開発されたプロトコルである．HoL ブロッキング問題は待ち時間問題とも呼ばれ，TCP パケットを連続で送信する際に，先頭パケットにエラーが発生した時に再送が完了するまで後続パケットを送信できない問題である．

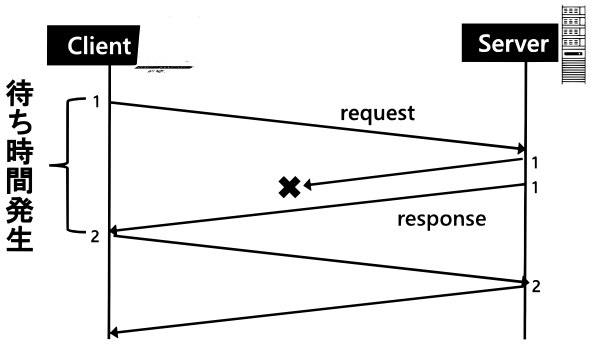

図 1　HoL ブロッキング問題

この問題に対して QUIC ではストリームという単位で並行処理させることで解決している．まず QUIC は TCP ではなく UDP ベースのためパケットを受信する順序に決まりがない．そして，複数ストリームにアプリケーションデータを格納して並行で送信することで，もしあるストリームでパケットロスが発生した際にも，別のストリームには影響せず運ぶことができ，待ち時間問題を回避できる．つまり独立したストリームの多重化によって HoL ブロッキングを引き起こさないようにしている．

また QUIC の上層のアプリケーション層で動く HTTP/3 は QUIC のストリームを使ってメッセージのやりとりを行う．つまり QUIC は HTTP のようにアプリケーションデータそのものを運ぶための基盤としても機能する．

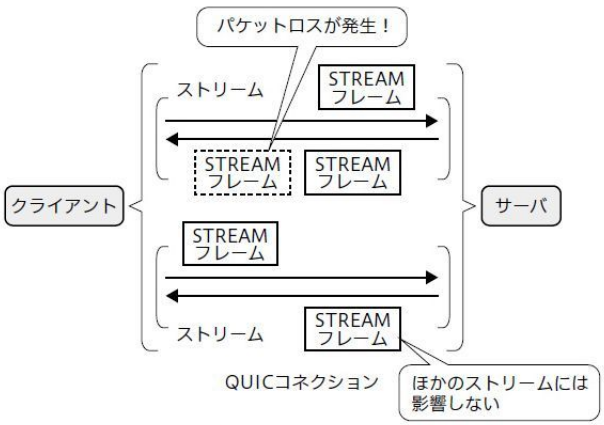

図 2　stream の様子 [3]

また，QUIC にはコネクションマイグレーションという機能がある．QUIC はコネクションをコネクション ID というパラメータで識別していて，IP アドレスが変わっても同じコネクションを引き続き使うことができる機能である．キャリア回線から Wi-Fi 回線へ切り替える際などに利用できる．コネクションマイグレーションを行った際は，新しいコネクション上では新しいコネクション ID を使うため，経路上の第三者が移行前と移行後の通信を関連付けることはできない [3].

2.2 0-RTT について

高速通信のための QUIC の機能の一つに 0-RTT(Zero Round-Trip Time) というものがある．0-RTT とは一度接続が確立している場合再接続時に RTT（往復時間）がゼロで通信できる機能である．図 3 のように通常 TCP と TLS で 5 回通信が必要だが，0-RTT を用いることで通信が 1 回で済むため大幅に通信開始までの時間を削減できる．仕組みとして，ハンドシェイクと同時にアプリケーションデータを格納した 0-RTT データを送ることで通信確立とデータ送信を同時に行っている．QUIC は TLS1.3 を統合しており，TLS1.3 の暗号的特性を利用し 0-RTT を利用できる．

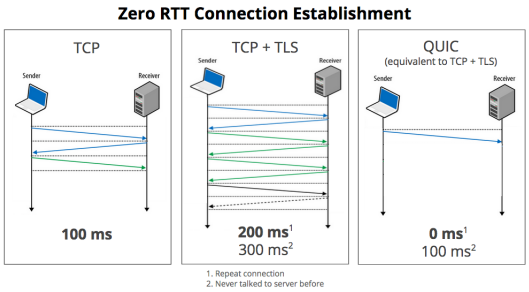

図 3　TCP と QUIC のハンドシェイク比較 [4]

2.3　QUIC のハンドシェイク

ここから 0-RTT を用いた QUIC のハンドシェイクを説明する．図 4 はハンドシェイクの手順である．

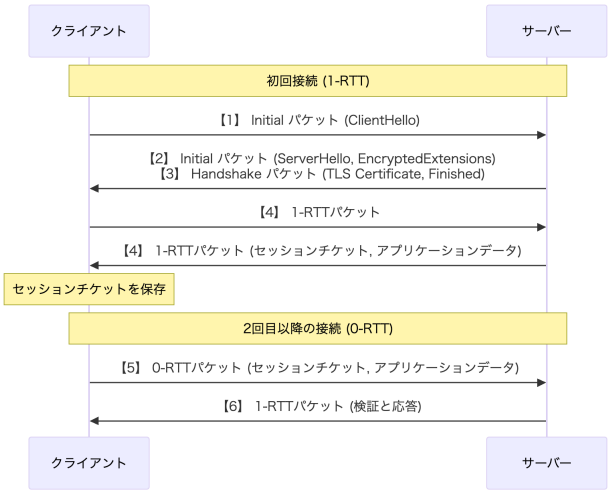

図 4　QUIC のハンドシェイクの手順

まず 1 回目のハンドシェイクについて解説する．

【手順 1】　クライアントは ClientHello を含む Initial パケットを送信して接続を開始する．

【手順 2】　サーバーは，ClientHello を受け取ると TLS の ServerHello と暗号化された拡張情報を含む Initial パケットで応答する．

【手順 3】　ハンドシェイクを完了させるために，TLS 証明書と Finished メッセージを含む Handshake パケットを送信する．

【手順 4】　ハンドシェイクが完了すると，サーバーはセッションチケットを含む 1-RTT パケットをクライアントに送り，クライアントはこのセッションチケットをローカルに保存する．

ここでセッションチケットとはセッションデータを相手サーバだけが復号できるように暗号化したものである．セッションデータには 1 つ前のハンドシェイクの情報が入っており，相手サーバー名や共通鍵暗号の種類，鍵の生成に使用する事前共有鍵 (Pre-Shared Key) などが含まれる．このセッションデータによって 0-RTT を可能にしている．セッションチケット以外にもセッションデータを DB に保存し，取り出すためのセッション ID を発行してクライアントに渡す方法もある．

続いて 2 回目の 0-RTT によるハンドシェイクを説明する．

【手順 5】　クライアントは保存されたセッションチケットを使い，ハンドシェイクを待つことなくアプリケーションデータ (0-RTT データ) を含む 0-RTT パケットをサーバーに送信する．

【手順 6】　サーバーは受信した 0-RTT パケットに含まれるセッションチケットを検証し，成功するとサーバーはアプリケーションデータに対する応答を含む 1-RTT パケットをクライアントに返す．

このように，QUIC では 0-RTT によってハンドシェイクとアプリケーションデータを同時に送信することでレイテンシが大幅に削減し，ウェブページの読み込みや API リクエストを高速化している．

3.　QUIC のセキュリティ上の課題とキャッシュポイズニング

3.1　QUIC のセキュリティ上の課題

QUIC は高速通信のための多くの機能を有しているが，それに伴い多くのセキュリティ上の課題が存在する．QUIC のセキュリティに関する survey 論文 [5] では，QUIC においてのいくつかのセキュリティ脅威を挙げている．

セキュリティ脅威の代表的なものとして，攻撃者がターゲットサーバーに QUIC 経由で大量のデータを送信してサービスを妨害する DDoS 攻撃がある．QUIC では UDP ベースであるため送信者の情報が受信者にほとんど提供しないことや，メッセージを TLS 暗号化し応答に TLS 証明書が含まれることで最初のメッセージよりも応答がはるかに大きくなるといった特徴がある．これらの要因から被害者の IP アドレスを模倣して，複数のサーバーに情報を要求するリフレクション型 DDoS 攻撃に対して特に脆弱である．これは大量の輻輳データを送信する増幅攻撃ともいえ，QUIC の DDoS 増幅率は最大 4.6 倍まで膨れ上がる [6]．QUIC における DDoS 攻撃の対処として QUIC パケットの許容数を制限するレート制限などがある [7]．また，QUICShield と呼ばれる DDoS 攻撃を迅速に検出する防御メカニズムも提案されている [8]．

QUIC のセキュリティに焦点を当てた論文 [9] では 0-RTT のセキュリティ課題について議論している．まず 0-RTT では鍵の前方秘匿性が提供されておらず，長期鍵が漏洩した場合に中間者攻撃などで過去の 0-RTT セッションのデータが解読される可能性がある．また，0-RTT は以前のセッションキーを用いて暗号化を行うため，以前に取得した有効なデータを再送信して DoS 攻撃を起こすリプレイ攻撃

に対しても脆弱である．2.2 で述べた通り 0-RTT はハンドシェイクを省略することで通信開始時の遅延を減らしているため，攻撃者によって再送信されたリクエストを正規のものと誤認しリクエストを処理してしまうことでリプレイ攻撃を可能にしてしまう．リプレイ攻撃には Source-Address Token(STK) リプレイ攻撃と Server Configuration(SCFG) リプレイ攻撃の 2 種類が存在する [6]．STK リプレイ攻撃はサーバーからクライアントに発行された STK を傍受して再利用し，サーバーに複数の不正な接続をおこなうことでリソース枯渇を引き起こす．SCFG リプレイ攻撃はクライアントによる初期接続リクエストからサーバーの公開設定を傍受し再利用することで，リプレイを気づかれずにクライアントからの初期キーを入手する．

これらの問題に対して鍵交換プロトコルの課題に取り組んだ論文 [10] にて前方秘匿性を提供するために PFS-KEM という新しい暗号プリミティブを提案している．PFS-KEM は暗号文が一度しか復号できないという特性を持っており，同じ暗号文で再送信を行えなくすることで 0-RTT プロトコルにおける前方秘匿性とリプレイ攻撃対策を実現している．また，STK リプレイ攻撃に対してはトークンを特定のセッションに暗号学的に紐づけることでセキュリティを確保し，悪用を防ぐためのレート制限を実装する方法も挙げられる [6]．

3.2　キャッシュポイズニング攻撃について

他の脅威として本研究で取り上げるキャッシュポイズニング攻撃が挙げられる．キャッシュポイズニング攻撃とは，キャッシュを不正に書き換えることでユーザーを偽のサイトに誘導するサイバー攻撃である．キャッシュポイズニング攻撃には DNS キャッシュポイズニング攻撃と Web(HTTP) キャッシュポイズニング攻撃が存在する．DNS キャッシュポイズニング攻撃は攻撃者が正規の DNS サーバーからの応答よりも先に偽の DNS 応答をキャッシュ DNS サーバーに送りつけ，不正な IP アドレスをキャッシュさせる方法である．攻撃を受けると正規のドメイン名を入力したにもかかわらず攻撃者が用意した偽の Web サイトに誘導されてしまい，フィッシング詐欺やマルウェアの感染が引き起こされる．

Web キャッシュポイズニング攻撃では攻撃者が Web サーバーと Web キャッシュの動作に攻撃を仕掛ける．Web キャッシュは Web コンテンツを一時的にサーバーに保存することで Web の応答速度や効率を上げることができるが，図 5 のように攻撃者がキャッシュを汚染しそれを利用者がアクセスすることで XSS や JavaScript インジェクションなどさまざまな攻撃につながる可能性がある．攻撃によってウェブサイトの改ざんが起きた場合，他ユーザーへ広範囲でのデータの漏洩などにつながるためプライバシー保護に対して脅威となりうる．本研究では HTTP/3 と関わりの深い Web キャッシュポイズニング攻撃を中心に調査を行った．

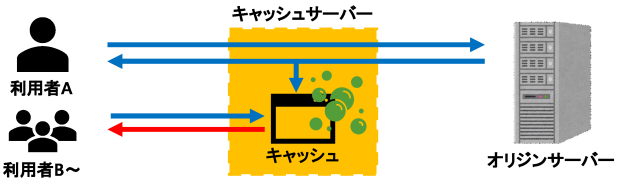

図 5　キャッシュポイズニング攻撃

Web キャッシュポイズニング攻撃の一例としてアンキャッシュキーを使う方法がある [11]．キャッシュキーは Host ヘッダなどのキャッシュに保存されたコンテンツを特定するための識別子であり，キャッシュキーでないリクエスト部分をアンキャッシュキーと呼ぶ．キャッシュされたレスポンスをユーザーに提供するかの判断にアンキャッシュキーは使われないため，アンキャッシュキー部分のペイロードに悪意のあるデータを挿入することで汚染しレスポンスを誘導する．

キャッシュポイズニング攻撃の歴史として 2008 年にダン・カミンスキー氏が DNS キャッシュポイズニングをより効率的に実行可能な新しい攻撃方法を発見したことから危険性が急激に高まったとされており，この攻撃はカミンスキー・アタックと呼ばれている [12]．カミンスキーアタックの特徴として目標に連続して繰り返し行うことができること，一部実装で既存のキャッシュが無効化されることがあり，これらが従来の攻撃と異なる点として脅威とされた．Web キャッシュポイズニングについては 2018 年に発表された論文 [13] にて広く知られるようになったとあり，比較的新しい攻撃手法である．Web キャッシュに対する攻撃の事例では，大手の米国決済代行会社においてキャッシュを不正に操作され，ウェブサイトの多くのページが XSS の脆弱性の影響を受けた．この脆弱性により，リクエストに含まれる X-Forwarded-Host ヘッダーの値が汚染されたままページ上に出力され，任意のスクリプトインジェクション攻撃が可能になっていたとある [14]．

Web キャッシュポイズニング攻撃を防ぐ方法としてキャッシュデータの整合性と真正性を保証する厳格な検証プロトコルの導入や，暗号化技術を活用してキャッシュの改ざんを防ぐことが効果的である [6]．適切なキャッシュ管理を行うことで攻撃のリスクを軽減し，QUIC のセキュリティを強化することができる．また，アンキャッシュキーによる悪用を防ぐために HTTP ヘッダーやクッキーからの入力利用を最小限にすることも有効である [15]．防衛策として最も確実な方法はキャッシュを無効化することであるが，キャッシュのメリットである Web サイトの読み込み時間の短縮が失われるため高速通信とのトレードオフとなる．

3.3 QUIC におけるキャッシュポイズニング攻撃

キャッシュの脆弱性の研究 [16] より，2024 年に開示された 30,000 件以上の脆弱性の中でキャッシュ関連の脆弱性がなりの部分を占めているという部分から，キャッシュの脆弱性については活発に研究されていることがわかる．論文にて Web キャッシュポイズニングは，最も一般的なキャッシュの脆弱性と書かれている．しかし，QUIC のセキュリティとプライバシーについての論文 [6] では，2024 年時点でキャッシュポイズニング攻撃が QUIC の分野ではまだ研究されておらず対策が不十分であること，QUIC のキャッシュメカニズムを悪用しデータ漏洩やシステム侵害を引き起こす可能性があることが書かれている．また，QUIC のセキュリティに関する survey 論文 [5] によると，QUIC におけるキャッシュポイズニング攻撃は脅威になるとしつつ先行研究はまだないとある．QUIC は多くのプロキシやロードバランサーに実装されているため，そういったインフラ環境でのキャッシュポイズニングの脅威の研究が行える可能性が書かれている．QUIC のセキュリティに焦点を当てた論文 [9] においても，QUIC に対するキャッシュポイズニング攻撃の評価はまだ行われておらず，QUIC には適切な対策手段がないことが書かれている．

QUIC におけるキャッシュポイズニングの先行研究が少ない理由として，0-RTT やコネクションマイグレーションなどの機能を有することによるプロトコル自体の複雑さやエンドツーエンドで暗号化していることからデバッグが難しいなどの要因がある [17]．プロトコルの難しさについて，HTTP/2 以前では下位層に TCP を使っていたが，HTTP/3 では TCP を使うことで起きる脆弱性を解決するため UDP 上で動作する QUIC プロトコルを用いている．つまり，HTTP/2 以前とは異なる新たな攻撃手法であるため既存の防御手法を使うのが難しいことも先行研究の少なさに拍車をかけている．

これらの先行研究より，プライバシー侵害という重要な問題でありながら QUIC の複雑さによって先行研究が十分にされていないことから，QUIC における Web(HTTP) キャッシュポイズニング攻撃について焦点を当てて研究を行った．

4. QUIC におけるキャッシュポイズニング攻撃の分析

4.1 キャッシュポイズニング攻撃の可能性

ここまでの調査をもとに QUIC 独自の機能を悪用しキャッシュポイズニング攻撃を行う際にどういった手法が考えられるか，0-RTT データへの攻撃，多重ストリームの悪用，コネクション ID 操作，この 3 つについてまず考察した．

1 つ目の 0-RTT データへの攻撃は 0-RTT にて接続を行った際の 0-RTT データに不正な HTTP リクエストヘッダや不正なペイロードを注入することで，キャッシュサーバーがそのリクエストに基づいて誤ったキャッシュを生成する可能性である．0-RTT は通信確立までの手順を大幅に省略できるが，3.1 で述べた通りリプレイ攻撃などに対して脆弱であるため，リプレイ攻撃と同じように過去のセッション情報を傍受し，その情報を用いて新たなリクエストを偽装しキャッシュを汚染できる可能性があると考えた．0-RTT では通信確立と同時にアプリケーションデータも同時に運ぶ．この 0-RTT データにキャッシュのキーとして利用される情報が含まれていて，かつそのデータがサーバー側で適切に検証されない場合，攻撃者がキャッシュを汚染し保存させて他のユーザーに提供させることが可能になると考える．

2 つ目の多重ストリームの悪用では，並行して処理しているストリーム間で順序の入れ替えを意図的に起こした場合に，キャッシュサーバーがこれを誤って解釈し，キャッシュポイズニングにつながる可能性である．しかし，QUIC は通信全体をエンドツーエンドで暗号化を行っているため，実際にネットワーク上でストリームを操作することは困難であること [17]，QUIC パケット自体も全て内部に TLS メッセージを組み込み暗号化していることから意図的にストリームの順序入れ替えを発生させてキャッシュポイズニングを起こさせるのは難しいのではないかと考えた．

3 つ目のコネクション ID 操作は攻撃者がコネクション ID を偽装したり意図的に再利用したりすることで，キャッシュサーバーが異なるユーザーからのリクエストを同一のコネクションと誤認しキャッシュを汚染する可能性である．しかし，QUIC ではセキュリティの仕組みとして 2.1 で述べた通りコネクションマイグレーションを行う際に別のコネクション ID を使うため偽装が難しいこと，トランスポート層の識別子であるコネクション ID が HTTP を扱うアプリケーション層のキャッシュにまで影響与える可能性は稀であると考え，この手法の可能性も低いと考えた．

それぞれを比較し考察した結果，本研究では 0-RTT の攻撃に焦点を当てその可能性についてより深く調査，研究を行った．

4.2 0-RTT への攻撃

実際に 0-RTT を悪用してキャッシュを汚染する際に，攻撃者がどういった偽のリクエストを送るかについて，不正なキャッシュ制御ヘッダ，偽装された Host ヘッダ，この二つを送信する方法を考えた．

まず，不正なキャッシュ制御ヘッダを送る場合である．キャッシュ制御ヘッダとは Cache-Control ヘッダのことで，ラウザーのキャッシュ動作を管理する HTTP ヘッダーである．このヘッダの設定に Cache-Control: public, max-age=3600 と共に，例えば X-Poison: true のようなカスタムヘッダを追加し汚染する方法である．しかし，キャッ

シュサーバーにてヘッダの正当性を厳しくチェックするため，キャッシュサーバーの環境次第になると考えた．

次に偽装された Host ヘッダを送る方法である．例えば正規のドメインとは異なる Host: malicious.com のような値を設定し送信することでキャッシュポイズニングを引き起こす．キャッシュサーバーが Host ヘッダをキャッシュキーの一部として適切に扱わない場合に効果的であると考える．現在はクラウドサービスが発展し多くのサーバにて 1 つの IP アドレスで複数のウェブサイトをホスティングする仮想ホスティングが基本であり，仮想ホスティングの識別とキャッシュキーの生成に URL だけでなく Host ヘッダを用いるため Host ヘッダによって誤ったキャッシュを生成できる可能性がある．

それぞれキャッシュサーバの環境による部分があるが，十分な検証が行われないキャッシュサーバに対しては攻撃のリスクが十分存在すると考えた．

5. 攻撃シナリオの考察

5.1 攻撃シナリオについて

第 4 章での分析をもとに，具体的な攻撃シナリオについてまとめる．攻撃のフローチャートは以下の通りである．

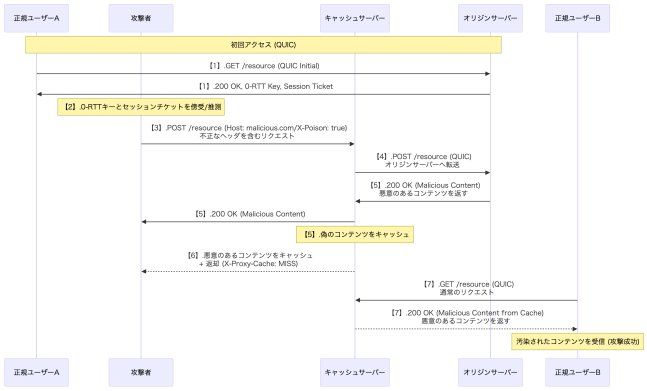

図 6 攻撃のフローチャート

【1】 正規ユーザー A が最初にウェブサイトのリソースに QUIC でアクセスする．この初回アクセス時にオリジンサーバーとの間で TLS ハンドシェイクが行われ，0-RTT キーとセッションチケットが生成される．

【2】 攻撃者は，この 0-RTT キーとセッションチケットを中間者攻撃などで傍受または推測する．

【3】 攻撃者は傍受/推測した情報を用いて，不正な Host ヘッダや不正なキャッシュ制御ヘッダを含むリクエストを 0-RTT データとしてキャッシュサーバーに送信する．

【4】 キャッシュサーバーが Host ヘッダをキャッシュキーの一部として適切に扱わない場合や，カスタムヘッダを適切に判断出来ないなどセキュリティの低い実装だった場合，不正なものと気づかず 0-RTT リクエストに基づいてオリジンサーバーに攻撃者の不正なヘッダが転送される．また，0-RTT データは完全なハンドシェイクを待たずヘッダ情報の入ったアプリケーションデータを送信できるため，攻撃が検知される前にオリジンサーバーに転送できる．

【5】 オリジンサーバーから返された偽のコンテンツ（偽のスクリプトや画像パスを含む HTML など）を自身のキャッシュに保存することでキャッシュの汚染が完了する．

【6】 キャッシュサーバーは汚染されたキャッシュに保存すると同時に，攻撃者クライアントに返却する．この際，攻撃者は返却されたコンテンツに含まれる X-Proxy-Cache: MISS や HIT ヘッダを見ることでキャッシュの状態を確認できる．

【7】 キャッシュが汚染された後，正規ユーザー B が通常のブラウザから同じ URL にアクセスした際に，正規ユーザー B のリクエストのキャッシュキーが攻撃者がキャッシュさせた悪意のあるコンテンツのキャッシュキーと一致すると判断し，キャッシュサーバーはキャッシュされた悪意のあるコンテンツを正規ユーザーに返す．

以上の流れで正規ユーザーは攻撃者の意図する不正なコンテンツを受け取り，キャッシュポイズニングが成功となる．

5.2 攻撃シナリオの見積もりと検証方法

攻撃シナリオの成功の有無には 0-RTT が早期処理を行っていることが重要だと考える．攻撃を阻止するセキュリティの仕組みとしてまずサーバー側の検証があり，検証ロジックを厳格に適用している場合に攻撃は阻止することができる．例としてセッションチケットの再利用が制限されていたり，0-RTT データ内のヘッダが正規の Host 名と一致しない場合にリクエストを破棄したりするような実装である．またキャッシュサーバー側ではキャッシュキーを生成する際に正規化の処理を行うため，キャッシュキー生成ロジックによって攻撃を阻止できる．他にも不正なヘッダやクエリパラメータを含むリクエストを検知し，ブロックする WAF(Web Application Firewall) などのシステムによる阻止も考えられる．

しかしこれらのセキュリティは 0-RTT にて早期に処理するために簡略化させることもあり，対象となるサーバーやキャッシュサーバーが 0-RTT リクエストをどのように処理するかによって成功率が変わると考えた．よって 0-RTT リクエストの検証が不十分な場合に攻撃が成功する可能性は十分にありうると考える．

攻撃シナリオの検証方法としてキャプチャソフトである Wireshark[18] を用いることで解析を行える．Wireshark で通信をキャプチャし，0-RTT データの構造，ヘッダの内

容，およびサーバーからのレスポンスを詳細に解析する．結果として改竄されたヘッダによってキャッシュされた不正なコンテンツが返されることを確認する．検証の比較対象として従来のプロトコル (HTTP/1.1, HTTP/2) における攻撃と解析を行い，HTTP/3 の 0-RTT と比較した際の攻撃の成功率，検知の難易度（ログ解析の容易性など），および影響度を評価することで攻撃シナリオの検証を詳細に行えると考える．

他の検証方法として qvis[19] というツールを使う手もある．qvis は輻輳制御変数などのプロトコル内部の状態を保存した qlog というデータを解析できる [17]．qvis によってストリームの多重化によるフローや輻輳制御などを視覚化できるため，通信のどこで攻撃が行われているのか等を詳しく見ることができ，攻撃の解析に役立つと考えた．

5.3 攻撃による影響

攻撃が成功した場合の影響として，まずコンテンツの改竄が挙げられる．例として正規のニュースサイトの記事が，悪意のあるページにすり替わるケースなどが考えられる．他の影響として情報漏洩の可能性もある．キャッシュに保存されるリソースには API から取得したユーザー情報などの動的なコンテンツも含まれることがあるため，キャッシュ汚染によって機密情報を意図的に漏出ささせることが可能になる．

攻撃者がキャッシュ汚染を行い，悪意のあるスクリプトを注入すると，そのスクリプトが他のユーザーの Cookie や認証情報を盗み出し，セッションを乗っ取るセッションハイジャックの危険性もある．セクションハイジャックによって攻撃者は被害者になりすましてサービスを不正に利用したり，個人情報にアクセスしたりすることが可能になる．

このようにキャッシュポイズニングが成功することで様々な攻撃を引き起こすことができてしまうため脅威になりうると考える．

5.4 攻撃の対策

攻撃の具体的な対策には 5.2 にて挙げた 0-RTT を考慮したサーバーの実装と検証やキャッシュポリシーなど各面から対策を講じることが必要だと考える．

まず 0-RTT データ処理の厳格化が考えられる．Host ヘッダや Cache-Control などの重要なヘッダが含まれるリクエストを受け取った際にキャッシュキー生成前に必ず検証する必要があると考える．また，0-RTT 利用によるデータ漏洩はセッションチケットの長期間利用も原因となるため，リプレイ攻撃の対策と同様に 0-RTT 通信を許可するセッションチケットの有効期限を短く設定したり，一度使用されたチケットを無効化したりすることで再利用を防ぐことも有効であると考える．キャッシュキーにおいても，URL エンコーディングの正規化や検証を行うことで攻撃者が意図的に不正な形式のリクエストを送信しても正規のキャッシュキーが生成されるようにすべきである．

キャッシュサーバーにおいて重要となるキャッシュポリシーの強化も有効だと考える．例としてログインページ，ユーザー設定画面，またはユーザーの個人情報を含む API レスポンスなどの密性の高い動的なコンテンツはキャッシュしないことや Cache-Control ヘッダを適切に設定することなどである．また，キャッシュされたレスポンスを使用するか判定するために使われる Vary ヘッダにて，必要最小限のキャッシュのみ行うようにすることも効果的と考える．

その他にも，3.2 にて解説したキャッシュポイズニングに対しての防衛策である検証プロトコルの導入や暗号化技術も有効である．

これら防御策のデメリットとして性能と利便性にトレードオフの影響を及ぼす可能性があることである．例えばセッションチケットの有効期限の短縮は，頻繁に新しいチケットを取得するためにフルハンドシェイクを行う必要があり，0-RTT によるパフォーマンス向上の機会が減少する．また，キャッシュ範囲の縮小もキャッシュサーバーがオリジンサーバーへのリクエストを転送する頻度が増え，遅延時間が伸びてしまう．QUIC による高速通信の恩恵を失わないようセキュリティとパフォーマンスのトレードオフを考慮してバランスを取ることが重要だと考える．

6. まとめと将来課題

6.1 まとめ

本研究では HTTP/3・QUIC におけるキャッシュポイズニングについて 0-RTT の観点から分析し，不正なヘッダを 0-RTT データ内に入れてリクエストを送り，汚染されたキャッシュを生成させてキャッシュポイズニングを行う攻撃シナリオを考察した．

研究の始めは，まず QUIC の高速通信の仕組みとそこに含まれるセキュリティ上の課題について先行研究で調査し，危険性のある攻撃ながらあまり研究が行われていない Web キャッシュポイズニング攻撃に焦点を当てた．そこから QUIC に対するキャッシュポイズニングの可能性について複数考え比較し，リプレイ攻撃の脅威と HTTP におけるヘッダの重要性から 0-RTT データ内のヘッダに脆弱性があるのではないかと仮説を立てた．その後具体的に不正な HTTP/3 ヘッダを注入することでキャッシュサーバーを汚染する攻撃シナリオを考察した．考察した攻撃シナリオの成功の有無には 0-RTT の早期処理とキャッシュサーバーの環境が関係すると考えた．攻撃シナリオによる影響として機密情報の漏洩などが考えられ，QUIC のセキュリティにおいて脅威になると考えた．攻撃の対策としてキャッシュ検証の厳格化が考えられ，具体的には 0-RTT を考慮

したサーバーの実装やキャッシュポリシーの強化，プロトコルの導入や暗号化技術などが考えられる．

本研究を通じて QUIC と HTTP/3 がウェブの高速化に貢献する一方で，0-RTT データや新しい HTTP/3 ヘッダの利用は従来のキャッシュポイズニング攻撃とは異なる新たな攻撃経路となり得ると考えた．

6.2 将来課題

本研究では攻撃シナリオの考察にとどまっており，実際に実証実験を行う段階まで到達できなかったため，今後の研究として 5.2 にて考察した攻撃シナリオの見積もりと検証方法に基づいた実験を行っていく予定である．実証実験を行う際には複数の環境を考え構築し比較していきたいと考えている．具体的には従来のプロトコルとの比較や DNS などの他のプロトコルに対するキャッシュポイズニング攻撃などで実装を行って検証を行っていきたい．また，QUIC の高速通信はハードウェアでも効果を発揮できると考えており，今回の先行研究の調査を活かしその実装方法とセキュリティについても検証を行いたい．

本研究では攻撃側のシナリオを考察したが，新たなセキュリティリスクとしてキャッシュポイズニング攻撃に対する防御策の検証と考察も実際に行っていく必要があると考える．QUIC のセキュリティとプライバシーについての論文 [6] にて提案されている検証プロトコルの導入や暗号化技術についてもより深く調査し研究し行っていきたい．キャッシュポイズニング攻撃の対処としてキャッシュを使わないことも有効ではあるが，QUIC のセキュリティ上の課題を解決し，キャッシュも活用して QUIC が目指す通信高速化をより追求していきたい．

参考文献

[1] ”W3Tech”: https://w3techs.com/technologies/details/ce-http3, (参照：2025.6.10),

[2] J. Iyengar, Ed. Fastly, M. Thomson, Ed. Mozilla: ”QUIC: A UDP-Based Multiplexed and Secure Transport”, https://datatracker.ietf.org/doc/html/rfc9000, Internet Engineering Task Force (IETF)(May 2021),

[3] 後藤ゆき: ”HTTP/3 入門 第 2 章 詳解 QUIC” https://portswigger.net/web-security/web-cache-poisoning , (参照：2025.8.12),

[4] Chromium Blog: ”A QUIC update on Google’s experimental transport”, https://blog.chromium.org/2015/04/a-quic-update-on-googles-experimental.html, (参照：2025.8.6),

[5] Y A Joarder, Carol Fung: ”A Survey on the Security Issues of QUIC”, 2022 6th Cyber Security in Networking Conference (CSNet)(October 2022),

[6] Y A Joarder, Carol Fung , Member, IEEE: ”Exploring QUIC Security and Privacy: A Comprehensive Survey on QUIC Security and Privacy Vulnerabilities, Threats, Attacks and Future Research Directions”, IEEE Transactions on Network and Service Management(September 2024),

[7] Akamai: ”QUIC フラッド DDoS 攻撃とは”, https://www.akamai.com/ja/glossary/what-is-a-quic-flood-ddos-attack, (参照：2025.8.8),

[8] Benjamin Teyssier, Y A Joarder, Carol Fung: ”QUIC-Shield: A Rapid Detection Mechanism Against QUIC-Flooding Attacks”, 2023 IEEE Virtual Conference on Communications (VCC)(November 2023), ,

[9] Efstratios Chatzoglou · Vasileios Kouliaridis · Georgios Karopoulos · Georgios Kambourakis: ”Revisiting QUIC attacks: a comprehensive review on QUIC security and a hands-on study”, International Journal of Information Security(December 2022),

[10] Felix G ¨ unther, Britta Hale, Tibor Jager, and Sebastian Lauer: ”0-RTT Key Exchange with Full Forward Secrecy”, Lecture Notes in Computer Science(April 2017),

[11] portswigger: ”Web cache poisoning”, https://portswigger.net/web-security/web-cache-poisoning, (参照：2025.8.8),

[12] 株式会社日本レジストリサービス (JPRS): ”新たなる DNS キャッシュポイズニングの脅威〜カミンスキー・アタックの出現〜”, https://jprs.jp/related-info/guide/topics-column/no9.html, (参照：2025.8.12),

[13] James Kettle: ”Practical Web Cache Poisoning”, https://portswigger.net/research/practical-web-cache-poisoning, (参照：2025.8.14),

[14] Seyed Ali Mirheidari, Matteo Golinelli, Kaan Onarlioglu, Engin Kirda, Bruno Crispo: ”Web Cache Deception Escalates!”, 1st USENIX Security Symposium (USENIX Security 22). USENIX Association, Boston (2022),

[15] James Kettle: ”Practical Web Cache Poisoning: Redefining ’Unexploitable’” YouTube, https://www.youtube.com/watch?v=j2RrmNxJZ5c, (August 2018),

[16] Fazel Mohammad Ali Pour: ”Cache Vulnerabilities: A Comprehensive Technical Analysis”, SSRN, http://dx.doi.org/10.2139/ssrn.5335918 (July 2025),

[17] Marx, Robin and Piraux, Maxime and Quax, Peter and Lamotte, Wim: ”Debugging QUIC and HTTP/3 with qlog and qvis”, ANRW ’20: Proceedings of the 2020 Applied Networking Research Workshop(July 2020),

[18] Wireshark Foundation: ”Wireshark,” Version 4.4, [Online] 入手先 ⟨https://www.wireshark.org/⟩, (参照：2025.7.28),

[19] qvis developers: ”qvis: QUIC Visualization Tools”, 入手先 ⟨https://github.com/quiclog/qvis⟩, (参照：2025.8.21),