キャッシュを包括するウェブセキュリティモデル

島本 隼人[1]　矢内 直人[1]　岡村 真吾[2]　藤原 融[1]

概要：ウェブの安全性解析として形式手法を用いた手法が提案されている。しかし、この既存研究のモデルには現実のウェブでは欠かせない概念であるキャッシュが含まれていない。実際に Browser Cache Poinsoning 攻撃といったキャッシュを用いた HTTPS に対する攻撃法が存在することから、キャッシュの概念はウェブの安全性解析には不可欠である。本研究では、キャッシュを包括したウェブセキュリティモデルを提案する。提案モデルではキャッシュの概念に加え、その運用を行う HTTP ヘッダや中継者の概念も新たに提案する。

キーワード：システム，キャッシュ，ウェブセキュリティモデル、セキュリティモデル、形式手法

Toward Formal Verification of Web Security with Cache

Hayato Shimamoto[1]　Naoto Yanai[1]　Shingo Okamura[2]　Toru Fujiwara[1]

Abstract: Cache is an important concept in a web system in order to provide them efficiently. Since a browser cache poisoning attack, where an adversary utilizes a web cache, has been proposed recently, analyzing the security under environments with cache is necessary. In this work, we propose a new security model including the cache. We also implement several parts of the model in Alloy.

Keywords: system, cache, web security model, security model, formal method

1. はじめに

1.1 研究背景

ウェブ上で個人情報を取り扱う機会の増加にともない，ウェブ通信の解析がより重要となっている．ウェブの解析はウェブモデルを用いた形式手法により自動化できることから，これらの研究が近年盛んである．しかし、もしモデルが包括している機能や要件定義が不十分である場合，適切な解析が行えない。そのため、使用するウェブモデルが適切であることが重要となる。

一般に、安全性要件の解析に使われるモデルはセキュリティモデルと呼ばれる。セキュリティモデルでは攻撃者の能力，運用する環境などの条件，許容できる被害レベルなどの安全性が定義されており，攻撃者の能力など各パラメータを数値化することで定量的な評価を可能とする．直観的には満たすべきセキュリティモデルを選択し，実装システムがそのモデルに含まれるかどうかを考察することで，実装モデルの安全性が確認できる．また，この解析を形式手法ツールを用い自動化することにより，安全性解析にかかる労力を大幅に削減できる．

この形式手法を用いた解析に向けて、Ahkawe ら [1] がセキュリティ機能を導入した初の汎用利用可能なモデルを提案している。しかしながら本研究では Ahkawe らのモデルにはキャッシュの概念が含まれていないことを指摘する．現在，キャッシュはその利便性から広く使われている技術であり，ウェブの安全性を考える上で欠かすことができない．事実，キャッシュを利用した攻撃法も近年の成果において報告されている [2]．以上のことから本研究はキャッシュの動作を包括するウェブセキュリティモデルの提案を目的とする。

[1] 大阪大学
Osaka University
[2] 奈良工業高等専門学校
National Institute of Technology, Nara College

1.2 貢献

研究で提案するウェブセキュリティモデルはウェブで使用するキャッシュの動作を、格納と再利用、再利用の可能性判定、レスポンスの検証の三点に分けて包括する。また、ウェブ上でキャッシュを搭載しうる存在としてクライアントとウェブサーバに加えて中継者があり、提案モデルではキャッシュの動作に必要な要素として導入している。さらに、キャッシュを実際に利用する際に必要となる HTTP ヘッダも新たに導入する。例えば、キャッシュの動作の指定に必要な Cache-Control ヘッダや有効期限の計算に必要となるレスポンスの生成時刻を記述する Date ヘッダである。

この提案モデルの一部を形式手法ツール Alloy を用いて実装している。また、Alloy の機能を用いて、実装を終えた部分については HTTP の仕様に従って正常に実装ができているかを確認している。

1.3 関連研究

Ahkawe らによる既存モデル [1] は本研究で提案するウェブセキュリティモデルのベースとなっている。このモデルは Bau らによる昨今の形式手法を用いた研究の調査論文 [3] においても取り上げられており、現在のウェブ環境においてよく使用される HTTP や DNS といった概念を包括し、現在のウェブの環境に最適なものである。しかしこのモデルにはキャッシュが包括されていない。また、Web のセキュリティに関連した研究としては、Barth ら [4] が SSL/TLS の安全性を調査している。本稿で対象とする攻撃者の能力は Barth らの研究で想定されている能力に相当する。また、本研究と同様に形式手法はウェブの様々な要素における安全性検証に用いられており、暗号アルゴリズムや SSL3.0、鍵交換プロトコル、認証プロトコルに対する検証が行われている [10–13]。

安全性に関する項目を含まない検証に対して形式手法は有効である。XML の機能を拡張する Web Services Enhancements(WSE) や Windows において通信 API として使用される Windows Communication Foundation(WCF)、Extensibe Markup Language(XML) の動作の検証 [6, 7] が行われている。また、ウェブに含まれる要素以外のシステムに対する安全性検証にも形式手法は利用されている。耐タンパー性を持つソフトウェアの動作を行うハードウェアの検証 [5] があり、形式手法が脆弱性とその対策に有効であることが示されている。同様に、ウェブにおける XML の機能を拡張する Web Services Enhancements(WSE) や Windows において通信 API として使用される Windows Communication Foundation(WCF)、Extensibe Markup Language(XML) に必要な安全性要求を形式手法を用いて発見した研究 [6,7] がある。また、OS カーネルの検証 [8] が形式手法を用いて行われており、形式手法を安全性証明として利用できることを示している。さらに、形式手法は AndroidOS の検証 [9] にも利用されている。この研究は OS のユーザ権限に着目した安全性検証を目指している。

2. Ahkawe らのモデルの問題

2.1 モデルの概要

Ahkawe らによる既存モデル [1] に包括されている概念は大きく三つである。一つ目はブラウザである。ブラウザはクライアントとしての基本的な動作と Cookie やスクリプトなどの動作が想定されている。二つ目はウェブサーバである。ウェブサーバでは HTTP レスポンスに対応する動作と DNS に対応する動作の二点に注目している。また、ネットワークの末端に位置し IP アドレスを保有するしていることが想定されている。三つ目はネットワークである。ネットワークには通信プロトコルや、それに必要な要素が包括される。既存モデルでは HTTP や HTTPS、DNS が包括されており、これらの運用に必要となる HTTP ヘッダや認証局（CA）なども含まれる。

また、既存モデルでは三種類の脅威モデルが想定されている。一つ目は web attacker と呼ばれる攻撃者である。web attacker は最も基本的な攻撃能力を持つ攻撃者であり、以下の能力を有する。

- 任意の HTTP リクエストの送信
- 管理するウェブサーバでの任意のレスポンスの生成
- 所有するドメインの正当なサーバ証明書の生成

二つ目の network attacker は web attacker の能力に加えて暗号化されていない通信の盗聴と改ざん、自己署名サーバ証明書を用いたなりすましの能力を持つ。三つ目の gadget attacker は web attacker の能力に加えてハイパーリンクや画像といった特定の形式のコンテンツを通信内容に挿入できる能力を持つ。

2.2 モデルの潜在的脆弱性

Ahkawe らによる既存モデル [1] では、キャッシュの概念が包括されていない。キャッシュはウェブサーバからクライアントへ送信されたレスポンスを保存する領域を持ち、そのレスポンスの再利用によって通信時間を短縮でき通信の効率化をおこなえる。このような利点からキャッシュは現在広く使用されている。

一方で、キャッシュを利用した攻撃法としてブラウザキャッシュ汚染攻撃（BCP 攻撃） [2] が知られている。BCP 攻撃は中間者攻撃の一つであり、ウェブサーバから送信されるレスポンスを中間者攻撃によって改ざんし、ブラウザに搭載されているキャッシュに保存させる。これにより、改ざんレスポンスがキャッシュからリロードされるため、ブラウザに悪影響を与えることが可能になる。たとえば、レスポンス内に悪質なサイトのリンクを追加するこ

とでブラウザを誘導したり、スクリプトを記述することによりブラウザに任意の動作を強制することができる。この攻撃法の特徴はキャッシュによるレスポンスの再利用機能を利用することにより、一度の中間者攻撃で長期に渡る悪影響が可能になる点である。また、この攻撃法は HTTPS に対する攻撃も可能である。この場合には、攻撃者は中間者攻撃を行う際に自己署名証明書を使用して正当な接続先になりすます必要がある。このなりすましに使用する自己署名証明書の安全性をブラウザでは確認できないため、ブラウザではユーザに対して認証警告が表示する。この認証警告をユーザが無視することが HTTPS に対する BCP 攻撃の条件である。

この BCP 攻撃のようにキャッシュを利用した攻撃法が存在すること、また、加えてキャッシュは一般に使用されているものであることから、ウェブの安全性解析を行う上でキャッシュは必要不可欠な概念である。したがって、キャッシュを導入したモデルを提案する必要がある。

2.3 キャッシュを包括するセキュリティモデルにむけて

キャッシュを導入したモデルの提案に伴って、合わせて必要となる概念がある。まず、中継者と呼ばれる概念である。HTTP において、キャッシュはクライアントとウェブサーバの他に中継者に搭載される。ここで中継者はネットワークの末端のクライアントとウェブサーバの経路上に存在するものであり、具体的にはプロキシやゲートウェイを指す。プロキシは通信経路を最適化することで通信効率を向上させ、また、クライアントの匿名性を保持するために使用される。ゲートウェイは内部のネットワークとウェブ（外部ネットワーク）を接続するサーバであり、ゲートウェイで翻訳を行うことにより異なるプロトコルの通信を可能にする。これらもキャッシュと同じくウェブで広く使用されている要素であるとともに、またキャッシュを使用する概念であることから、キャッシュを導入したモデルを提案する際にはこの中継者の概念も追加する必要がある。モデルにはこれらに加えて、キャッシュおよび中継者の動作に使用されるヘッダが新たに必要となる。レスポンスをキャッシュに保存する際に適応するオプションやキャッシュ内での有効期限はレスポンス内のヘッダで指定する。このヘッダはキャッシュの運用にのみ使用されるヘッダであり、提案モデルには導入しておく必要がある。同様に、中継者の動作に必要なヘッダについても追加する必要がある。

3. 提案モデルの設計

本研究で提案するウェブセキュリティモデルの特徴はキャッシュ、ヘッダ、中継者の三つに分けて考えることができる。

3.1 キャッシュの定義

提案モデルではキャッシュの基本的な運用に対する動作を包括する。ここで包括する動作は以下の三つに分類できる。

3.1.1 レスポンスの格納と再利用

キャッシュは以下の条件すべてを満たしている場合にレスポンスを保存することができる。

- キャッシュを禁止するオプションが含まれていない
- 有効期限を計算可能、もしくは、キャッシュを許可する記述がある

また、リクエストが要求している URI をもつレスポンスを保持しているキャッシュは、以下のいずれかに該当する場合にそのレスポンスを再利用してリクエストに返送することができる。

- レスポンスが有効期限内
- 有効期限を過ぎているレスポンスの再利用が許可されている
- レスポンスが検証済み

3.1.2 キャッシュの検証

キャッシュにおける検証とは、キャッシュ内のレスポンスに含まれているコンテンツがオリジンサーバ内の最新のコンテンツと一致するかを確認する動作である。検証が必要になった場合、キャッシュはオリジンサーバに対して “条件付きリクエスト” を送信する。条件付きリクエストとはヘッダで指定した条件を満たしている場合に完全なレスポンスを返すリクエストである。この条件にはコンテンツの最終更新時間、もしくは、レスポンス内の Etag ヘッダから得られる固有値を使用する。最終更新時刻を用いる場合には If-Modified-Since ヘッダを、固有値を使用する場合には If-None-Match ヘッダを使用した条件付きリクエストを生成する。通信において問題が発生しなかった場合、このリクエストに対するレスポンスの状態コードは 200(OK) もしくは 304(Not Modified) が想定される。200(OK) の場合、キャッシュはキャッシュ内のレスポンスを破棄し、新たに取得したレスポンスを使用する。このとき格納可能条件を満たしていれば、新しいレスポンスをキャッシュしてもよい。304(Not Modified) であった場合、キャッシュ内のレスポンスのヘッダを今取得したレスポンスのヘッダに置き換えて再利用する。このヘッダの置換によりレスポンスの有効期限を延長する。このヘッダの置換時に Waring ヘッダが存在する場合、1xx および 2xx の値は除去する。また、同じ URI に対するレスポンスが同じキャッシュに複数存在する場合、検証結果として送信されるレスポンスに記述されている検証子に該当するレスポンスのヘッダを置換して再利用する。複数のレスポンスに該当する場合には、そのレスポンスのうち最新のレスポンスを採用する。

3.1.3 キャッシュの有効期限

最後は、レスポンスの有効期間の判定である。この判定

にはレスポンスの有効期限と生成されてからの経過時間が必要となる。まず、キャッシュは以下の値を有効期間として採用する。複数個存在する場合にはより上位に書かれている値を採用する。

(1) キャッシュが共有キャッシュであり s-maxage が存在する場合、s-maxage の値
(2) max-age が存在する場合、max-age の値
(3) Expires ヘッダが存在する場合、(Expires の値 - Date ヘッダの値) で求められる値

もし、同項目に複数個の値が示されている場合、例えば、max-age に異なる値が 2 つ以上記述されている場合には、そのレスポンスは常に有効期限切れとして扱う。また、上記のどの値も得られなかった場合、キャッシュは任意の期限を設定する。この期限の設定では Last-Modified ヘッダの値を用いて計算することが多い。

次に、キャッシュが経過時間を計算するには以下の値が必要となる。

$$
\begin{aligned}
age &= \text{Age ヘッダの値} \\
date &= \text{Date ヘッダ値} \\
current &= \text{現在時刻} \\
reqtime &= \text{リクエストの受信時刻} \\
restime &= \text{格納レスポンスの受信時刻}
\end{aligned}
$$

これらの値を用いて、経過時間 T_{age} は以下の手順で計算される。

$$
\begin{aligned}
T_{apparent} &= max(0,\ restime - date) \\
T_{corrected} &= age + (restime - reqtime) \\
T_{initial} &= max(T_{apparent},\ T_{corrected}) \\
T_{visit} &= current - restime \\
T_{age} &= T_{initial} + T_{visit}
\end{aligned}
$$

3.2 ヘッダの定義

（節をわける）

提案モデルでは、キャッシュおよび中継者の動作に必要となるヘッダを新たに包括する。表 1 にその一覧を示す。

ヘッダフィールドはヘッダを用途と役割によって分類する。Ahkawe らによる既存モデル [1] では、HTTPRequestHeader と HTTPResponseHeader が含まれており、これらに加えて提案モデルでは HTTPGeneralHeader と HTTPEntity ヘッダを追加している。この四種類の分類は HTTP の仕様においても用いられている。HTTPRequestHeader と HTTPResponseHeader はそれぞれ、リクエストもしくはレスポンスでのみ使用可能なヘッダが属する。これに対して、HTTPGeneralHeader と HTTPEntityHeader はリクエストとレスポンスのどちらにも使用可能なヘッダが属しており、この二つの違いは属するヘッダの役割である。HTTPGeneralHeader に含まれるヘッダはリクエストやレスポンスそのものの情報を表す。例えば、レスポンスのサーバにおける生成日時を記録するために用いられる。一方で、HTTPEntityHeader に属するヘッダはリクエストやレスポンスに含まれているコンテンツの情報を表す。例えば、レスポンスで送信されているファイルの最終更新日時をクライアントに伝達するといった役割を担う。

表 1 提案モデルで追加する HTTP ヘッダ

HTTP ヘッダフィールド	HTTP ヘッダ
HTTPRequestHeader	IfModifiedSinceHeader
	IfNoneMatchHeader
	PragmaHeader
HTTPResponseHeader	AgeHeader
	EtagHeader
	LastModifiedHeader
HTTPGeneralHeader	CacheControlHeader
	ConnectionHeader
	DateHeader
	WarningHeader
HTTPEntityHeader	ExpiresHeader

また、表 1 に含まれているヘッダのうち、特に重要な役割をもつ一部のヘッダを説明する。一つ目として Cache-Control ヘッダが考えられる。このヘッダは様々なオプションの記述によって、キャッシュの動作を指定することができる点で、キャッシュの動作に強く影響を与えるヘッダであるという点で重要である。使用するオプションの一例として max-age があり、非負整数を記述することでレスポンスがキャッシュに格納された際の有効期限を指定することができる。また、キャッシュへの保存を許可しない no-store、検証を行わないレスポンスの再利用を許可しない no-cache といったキャッシュの動作に制限を与えるオプションも存在する。

次に、キャッシュ内のレスポンスが再利用可能か判定するために必要なヘッダについて考える。再利用可能判定では、レスポンスの有効期限と生成からの経過時間が必要となり、経過時間の計算は 3.1 節において前述した計算式を用いる。したがって、この計算で用いる Date ヘッダ、Age ヘッダはレスポンスの再利用可能判定に必須であり重要なヘッダである。Date ヘッダはレスポンスのオリジンサーバにおける生成時刻、Age ヘッダはレスポンスの生成からの経過時間を表す。

最後にキャッシュの検証機能に必要なヘッダを考える。検証の動作はコンテンツの最終更新時間を用いる手法と、固有値を用いる手法の二種類が存在する。最終更新時間を用いる検証には、最終更新時間を記述する LastModified ヘッダと、記述されている時刻以降の更新が行われている場合にコンテンツの再送信を求める If-Modified-Since ヘッダが必須である。また、固有値を用いる検証には、固有

値を伝達する Etag ヘッダと、記述されている固有値と最新のコンテンツの固有値が異なる場合に再送信を求める If-None-Match ヘッダが必要である。

提案モデル内には、役割が重複しているヘッダも存在している。例えば、Expires ヘッダはレスポンスの有効期限を、Pragma ヘッダは no-cache のオプションを表すことができるが、これらは Cache-Control ヘッダを使用することでも表現可能である。これは、Cache-Control ヘッダが HTTP/1.1 で新しく追加されたものであるからであり、HTTP/1.1 においては Expires ヘッダ、および、Pragma ヘッダの使用は推奨されていない。この二つのヘッダは、HTTP/1.1 が動作していない環境のみの使用が想定されていしかし、既存モデルは HTTP/1.0 と HTTP/1.1 の両方を含む環境を想定しているため、提案モデルにおいてもこの想定環境を引き継ぐことを目的としてこの二つのヘッダを包括している。

3.3 ウェブ要素の定義

提案モデルでは、包括しているウェブの構成要素を図 1 のように構成している。図 1 において、濃く表示している要素は提案モデルで新たに追加した概念であり、矢印はクラスの継承構造を表す。

図 1 の一番上に存在する NetworkEndpoint はネットワークの末端に存在する要素の概念を表し、HTTPConfirmist は HTTP の仕様に従う動作をする概念を表す。既存モデルでは、クライアントとウェブサーバがこの概念に含まれており、この二つと並列に中継者の概念 (HTTPIntermediate) を追加している。さらに中継者はプロキシ (HTTPProxy) とゲートウェイ (HTTPGateway) の概念を内包している。

HTTP/1.1 においてプロキシは通過するコンテンツへの編集と通信経路の設定の二つの機能を持つが、これらの機能はキャッシュの機能に大きな影響は与えない。提案モデルはキャッシュの動作に注目した安全性検証を目的としているため、プロキシを通信経路上に存在するキャッシュを搭載する要素とみなす。

ゲートウェイは外部ネットワークから HTTP を用いてアクセスされる場合、ウェブサーバとして認識され同様の動作を行う。このことから、提案モデルではゲートウェイの内部ネットワークにおける動作は考慮せず、ウェブサーバとみなして設計する。

4. モデル実装

提案モデルの実装は形式手法ツールである Alloy を用いて進めている。Alloy はモデルを与えて実行することで、そのモデルがとり得る状態をすべて出力する。この出力された状態が安全であるかを確認することで、システムの安全性を証明することができると同時に、潜在的な危険性の発見が可能である。また、出力する状態を限定する条件を

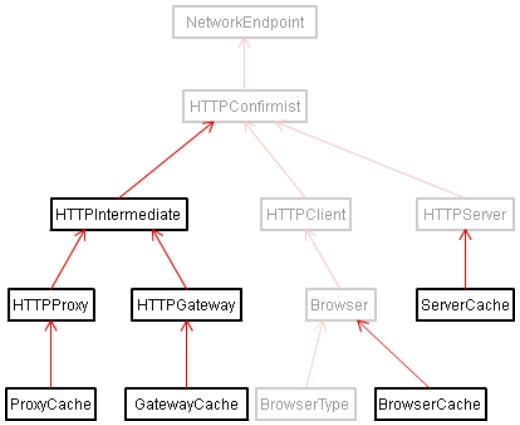

図 1 提案モデルに含まれるウェブアーキテクチャ

設定することができ、システムがとってはならない動作を条件として設定することで、その状態に至る原因の発見にも役立つ。Ahkawe らによる既存モデル [1] はこの Alloy を用いて実装が行われており、提案モデルはこれを拡張して実装を進める。

実装を行う項目と実装を終えている項目を表 2 に示す。また、モデルにおける実装済みの項目の構造は図 2 のように表される。図 2 内の黒の矢印は継承関係を表し、その他の色は集合ごとの関係性を表す。例えば、General ヘッダフィールドに属している Cache-Control ヘッダは、モデル内では HTTPGeneralHeader を CacheControlHeader の性質を継承していることが表現されている。また、Cache-Control ヘッダとそのオプションを表す CacheOption は options と関連付けられている。

表 2 実装項目

実装項目	実装済み
レスポンスの格納	○
レスポンスの再利用	○
レスポンスの検証機能	-
中継者	-
ヘッダ	△（一部）

提案モデル内で整数値は Int クラスを関連付けることで表現できる。例えば、CacheControl ヘッダの max-age のようないくつかのオプションには整数値を用いるものが存在し、このようなオプションに Int クラスが関連付けられていることが図 2 に示している。同様に、日時を表す値を必要とする Date ヘッダにも Int を関連付けている。本来、日時を表す値と整数値は異なるものであるが、HTTP において日時を表す値を用いる際には対応する整数値に変換して扱うため、Int クラスを関連付けることは HTTP の仕様に反してはいない。

ここで、Cache-Control ヘッダの記述を以下に示す。

```
1: sig CacheControlHeader extends HTTPGeneralHeader
     {options : set CacheOption}
```

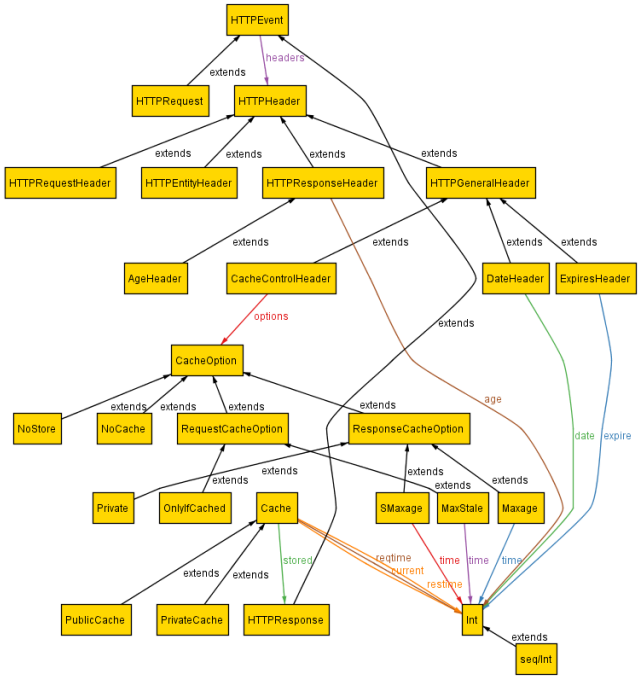

図 2　実装中の提案モデル

```
2: abstract sig CacheOption{}
3: abstract sig RequestCacheOption
      extends CacheOption{}
4: abstract sig ResponseCacheOption
      extends CacheOption{}
5: sig NoCache,NoStore extends CacheOption{}
6: sig OnlyIfCached extends RequestCacheOption{}
7: sig MaxStale extends RequestCacheOption
      {time: one Int}{time > 0}
8: sig Private extends ResponseCacheOption{}
9: sig Maxage,SMaxage extends ResponseCacheOption
      {time: one Int}{time > 0}
```

Cache-Control ヘッダは属する HTTPGeneralHeader の性質を継承しており、オプションを示す CacheOption の集合を有する。各オプションはこの CacheOption の性質を継承し、整数を持つオプションのみ整数を有している。

これらのオプションとその他のヘッダを用いて、レスポンスの有効期限の条件を表現している。以下にその記述を示す。

```
1: let A = HTTPResponse.headers.age,
   D = HTTPResponse.headers.date |
2:    let apparent=(restime.minus[D]>0 implies
      restime.minus[D] else 0), corrected =
      A.plus[restime.minus[reqtime]] |
3:       let initial=(apparent>corrected
         implies apparent else corrected) |
4:          Maxage.time.minus[initial.plus
      [current.minus[restime]]]>0
```

このコードは、PrivateCache における max-age の値を有効期限に採用する際に満たしているべき条件、つまり、3.1.3 節における計算における値が max-age の値よりも小さいことを示している。

また、表 2 の未実装の項目は、現在実装を進めており今後の課題である。

5.　実装モデルの正当性

提案モデルのヘッダ定義が正当であることを確認する。前述した通り実装モデルを与えて Alloy を動作させると、そのモデルを満たす状態が出力される。本章では、出力された状態が HTTP の仕様を満たしていることを確認することで、正当性を確認する。

5.1　レスポンスの格納と再利用

図 3 が表す状態 a では、ある HTTP リクエストに対して個別キャッシュに格納されているレスポンスを再利用し応答することが可能であることが示されている。

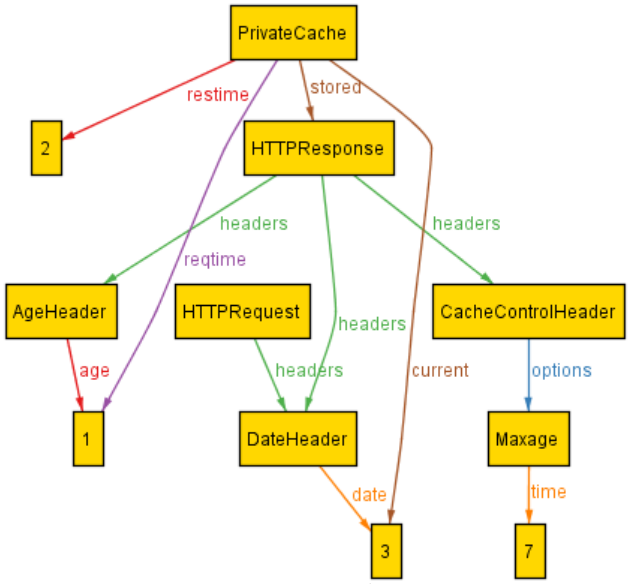

図 3　状態 a

図 3 より、この HTTP リクエストのヘッダは以下のように想定されている。

```
Date: 3
```

同様に、HTTP レスポンスは以下のヘッダが想定されている。

```
Date: 3
Age: 1
Cache-Control: max-age=7
```

ここで、このレスポンスのキャッシュへの格納、およびこの状況における再利用が正当であるかを考察する。まず、

このレスポンスにはキャッシュを禁止する記述はなく、かつ、有効期限を明記する Cache-Control ヘッダの max-age オプションが指定されているため、3.1.1 節のキャッシュへの格納条件を満たしている。次に、3.1.3 節のキャッシュの再利用条件について考察する。レスポンスのヘッダに有効期限を過ぎているレスポンスの再利用を許可する記述はなく、キャッシュの検証も行っていないため、レスポンスが有効期限内である必要があり図 3 の情報から判定を行う。有効期限は max-age の値が採用され 7 となり、レスポンスの経過時間 T_{age} を 3.1.3 節の計算式を用いて求める。

$$age = 1,\ date = 3,\ current = 3$$
$$reqtime = 1,\ restime = 2$$
$$T_{apparent} = max(0,\ restime - date) = 0$$
$$T_{corrected} = age + (restime - reqtime) = 2$$
$$T_{initial} = max(T_{apparent},\ T_{corrected}) = 2$$
$$T_{visit} = current - restime = 1$$
$$T_{age} = T_{initial} + T_{visit} = 3$$

以上より経過時間は有効期限内となり、レスポンスは再利用可能条件を満たしている。したがって、状態 a は HTTP の仕様を満たしている。

5.2 Private オプション

以下のコマンドにより、Private オプションが記述されている際に共有キャッシュにレスポンスが格納されていないかを確認できる。実行の結果、共有キャッシュにレスポンスが格納されている反例が存在しなかったため、Private オプションは正常に動作している。

```
assert checkPrivate{
  #Private>0 implies #PublicCache.stored=0
}
check checkPrivate for 10
```

5.3 No-store オプション

以下のコマンドにより、No-store オプションが記述されている際にキャッシュにレスポンスが格納されていないかを確認できる。実行の結果、キャッシュにレスポンスが格納されている反例が存在しなかったため、No-store オプションは正常に動作している。

```
assert checkNoStore{
  #NoStore>0 implies #Cache.stored=0
}
check checkNoStore for 10
```

6. 結論

本研究では，Ahkawe らの既存のウェブセキュリティモデル [1] に不足点があることを受けて、新たにキャッシュを含むセキュリティモデルを提案した。主にキャッシュの動作，キャッシュを搭載するウェブの構成要素，キャッシュの動作に使用するヘッダについて HTTP/1.1 及び HTTP/1.0 の仕様に従ってモデルを作成した．また，提案モデルの実装を行っており、実装を終えている機能についての正当性の確認を行った．その結果、レスポンスの格納や再利用、その条件判定について正常に動作していることを確認した。

本研究の今後の課題は、実装を終えていない要素の実装である。また、この提案モデルの実装後は、現在研究が進められているプロトコルである HTTP Strict Transport Security (HSTS) [14] および Public Key Pinning for HTTP (HPKP) [15] を包括したモデルの提案を考えている。これら二つのプロトコルは暗号通信のプロトコルとして現在広く使用されている HTTPS に含まれる脆弱性を補ったものであり、動作の実装にキャッシュを使用しているため提案モデルとの関連性は高い。HSTS と HPKP の運用には専用のヘッダを必要としているため、その専用ヘッダの定義も行う。

謝辞 本研究の一部は，JSPS 科研費 16K16065 の助成を受けている．

参考文献

[1] Akhawe, D., Barth, A., Lam, P. E., Mitchell, J. and Song, D.: Towards a Formal Foundation of Web Security, *IEEE Computer Security Foundations Symposium*, pp. 290–304 (2010).

[2] Y.Jia, Yue, C., Xinshu, D., Prateek, S., Jian, M. and Zhenkai, L.: Man-in-the-browser-cache: Persisting HTTPS attacks via browser cache poisoning, *Computers and Security*, Vol. 55, pp. 62–80 (2015).

[3] Bau, J. and Mitchell, J. C.: Security Modeling and Analysis, *IEEE Symposium on Security and Privacy*, Vol. 9, pp. 18–25 (2011).

[4] Barth, A., Jackson, C. and Mitchell, J. C.: Securing Frame Communication in Browsers, *Communications of the ACM*, Vol. 52, pp. 83–91 (2009).

[5] Lie, D., Mitchell, J., Thekkath, C. A. and Horowitz, M.: Specifying and Verifying Hardware for Tamper-Resistant Software, *Security and Privacy*, pp. 166–177 (2003).

[6] Bhargavan, K., Fournet, C. and Gordon, A. D.: Verified reference implementations of WS-Security protocols, *Lecture Notes in Computer Science*, Vol. 4184, pp. 88–106 (2006).

[7] Gordon, A. D. and Pucella, R.: Validating a web service security abstraction by typing, *Formal Aspects of Computing*, Vol. 17, pp. 277–318 (2005).

[8] Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H. and Winwood, S.: seL4: Formal Verification of an OS Kernel, *Symposium on Operating Systems Principles*, pp. 207–

220 (2009).

[9] Shin, W., Kiyomoto, S., Fukushima, K. and Tanaka, T.: Towards Formal Analysis of the Permission-Based Security Model for Android, *International Conference on Wireless and Mobile Communications*, pp. 87–92 (2009).

[10] Mitchell, J. C., Mitchell, M. and Stern, U.: Automated analysis of cryptographic protocols using Murφ, *IEEE Symposium on Security and Privacy*, pp. 141–151 (1997).

[11] Mitchell, J. C., Shmatikov, V. and Stern, U.: Finite-State Analysis of SSL 3.0., *USENIX Security Symposium* (1998).

[12] Roscoe, A. W.: Modelling and verifying key-exchange protocols using CSP and FDR, *IEEE Computer Security Foundations Symposium*, pp. 98–107 (1995).

[13] Song, D. X.: Athena: a new efficient automatic checker for security protocol analysis, *IEEE Computer Security Foundations Symposium*, pp. 192–202 (1999).

[14] Hodges, J., Jackson, C. and Barth, A.: HTTP Strict Transport Security (HSTS), *Request for Comments*, No. 6797, (online), available from ⟨http://www.ietf.org/rfc/rfc6797.txt⟩ (2012).

[15] Evans, C., Palmer, C. and Sleevi, R.: Public Key Pinning Extension for HTTP, *Request for Comments*, No. 7469, (online), available from ⟨http://www.ietf.org/rfc/rfc7469.txt⟩ (2015).