L-035

OpenSSL キャッシュタイミング攻撃の実現性について
Analysis of the Possibility of cache timing attacks against OpenSSL

長谷川 淳† 渡辺 優平† 森井 昌克†
Jun Hasegawa Yuhei Watanabe Masakatu Morii

1. はじめに

現在，我々はインターネットによるオンラインサービスによって様々な利便性を享受している．ネットショッピングやインターネットバンキングといった商業的なものから確定申告といった公的機関による手続きまで，幅広い分野がオンラインサービスを提供している．こういったサービスを利用するとき，懸念されるのは個人情報の安全性である．これを保障しているのが暗号技術と，それを応用したセキュリティプロトコルである．セキュリティプロトコルの代表としてウェブやメールに幅広く利用されているものに SSL / TLS がある．SSL(Secure Sockets Layer)[1]は Netscape Communications 社によって設計された．その後継として TLS(Transport Layer Security)[2]が開発された．現在利用されている SSL / TLS は主に TLSv1.2, TLSv1.1, TLSv1.0, SSLv3.0 である．これらのプロトコルをサポートしているサイトは 2015 年 6 月時点でそれぞれ 62.1%, 59.6%, 99.5%, 37.7%となっている[11].

これらのプロトコルに対し，鍵および平文を回復する攻撃が相次いで報告されている．Bernstein は 2005 年に暗号化に AES を用いた場合，キャッシュによる時間差を利用し，鍵を取得するキャッシュタイミング攻撃[3]を提案した．2011 年には SSLv3.0 および TLSv1.0 に対する攻撃として BEAST(Browser Exploit Against SSL / TLS)[4]が提案された．2012 年の CRIME(Compression Ratio Info-leak Made Easy/Compression Ratio Info-leak Mass Exploitation)[5]，2013 年の BREACH(Browser Reconnaissance and Exfiltration via Adaptive Compression of Hypertext)[6]はともに SSL / TLS のデータ圧縮機能を利用した攻撃として報告された．2013 年には MAC の計算速度の違いを利用した Lucky13[7]が提案された．2014 年に発表された POODLE(Padding Oracle On Downgraded Legacy Encryption)[8]は，SSLv3.0 のプロトコル上の欠陥を利用した攻撃である．SSL / TLS においてブロック暗号を用いて暗号化を行うとき，平文のサイズをブロック長の倍数にあわせるためにパディングという処理を行う．SSLv3.0 の仕様では，パディングしたバイトの中身は確認しない．SSLv3.0 は TLS 以前に開発されたプロトコルであるが，互換性を維持するために数多くの機器でサポートが続いている．攻撃者はこの状況を悪用し通信を改竄することで，SSLv3.0 を用いた通信を強制することができる．またブロック暗号の特性によりブロックの境界線を移動させて，攻撃を効率よく行えるようにする．結果として攻撃者は暗号化された HTTP リクエストを復号することができる．POODLE はプロトコルの仕様そのものを利用した攻撃であり，SSLv3.0 においては現実的な対抗策は無いと言われている．

† 神戸大学, Kobe University

本稿ではこれらの攻撃のうち，Bernstein のキャッシュタイミング攻撃のクラウド環境下での実現性について検証する．クラウド環境とはソフトウェアやハードウェア資源をネットワークを通じて利用可能な環境の事であり，コストや保守管理の点で優れているため，企業などで導入が進んでいる．このクラウド環境を支える仮想化技術は，一台の PC 内に複数の VM を動作させて処理を行っている，この特性を利用し，Bernstein の攻撃がより現実的に実行できないかを検証する．

2. SSL / TLS

本章では SSL / TLS の仕様について述べ，暗号文を解読する攻撃である BEAST および POODLE について説明する．

2.1 SSL / TLS の仕様

SSL / TLS はウェブ上でのなりすましや盗聴，改竄などの脅威への対抗策を実現するためのセキュリティプロトコルの一つである．これらの要件は機密性，完全性，真正性とも言われ，SSL / TLS は全てを満たすように設計および運用されている．ウェブ上でオンラインショップなどを通じてクレジットカード番号などの秘密情報をやりとりするとき，ウェブに利用する HTTP だけでは平文で情報がやり取りされてしまう．そこで HTTP を通じても安全に通信するプロトコルとして SSL / TLS が利用されている．SSL / TLS を HTTP と共に用いた場合，それらのプロトコルは HTTPS(HTTP over SSL)と呼ばれる．SSL / TLS を用いて暗号化したレコードはヘッダブロックと暗号化データブロックに分類される．ヘッダブロックは暗号化されておらず，暗号化データブロックに関するメタデータが格納されている．

2.2 POODLE

POODLE(Padding Oracle On Downgraded Legacy Encryption) は Bodo Möller, Thai Duong, Krzysztof Kotowicz によって 2014 年に提案された SSLv3.0 に対する攻撃手法であり，CBC モードのブロック暗号を用いたとき，中間者攻撃によって暗号化された HTTP リクエストの解読が可能となる．HTTP リクエストのヘッダを復号することで Cookie 情報を取得することもでき，BEAST 同様セッションハイジャックにつながる恐れがある．

SSLv3.0 は TLS と比べ古いプロトコルであり優先して使われることは無いが，多くのサイトでは互換性のために SSLv3.0 のサポートを続けている．そしてクライアントおよびサーバどちらかが SSLv3.0 しかサポートしていない時，SSLv3.0 を使う．POODLE はこの性質を利用し，両者が TLS をサポートしていても強制的に SSLv3.0 を使用させる．これはバージョンロールバック攻撃と呼ばれる．攻撃者に必要な条件・環境は以下の通りである．

- クライアントの PC を通して任意のリクエストライン，ボディを記述した HTTP リクエストが送信可能である．
- 全てのパケットの盗聴および改竄が可能である．
- 暗号化には CBC モードのブロック暗号が用いられる．
- サーバおよびクライアントは SSLv3.0 をサポートしている．

3. キャッシュタイミング攻撃

本章では既存のキャッシュタイミング攻撃について説明する．まずこれまでの関連研究について述べ，次に Bernstein によるキャッシュタイミング攻撃実装した結果を示す．

3.1 関連研究

暗号化にキャッシュを用いる場合に発生する時間差を用いたキャッシュタイミング攻撃はこれまで数多く報告されている．Kelsey らは 1998 にキャッシュのヒット率をもとに攻撃を行う Side Channel Cryptanalysis of Product Ciphers[13] を提案した．2005 年，Bernstein は既知鍵と未知鍵で暗号化にかかる時間を比べ，鍵回復を行うキャッシュタイミング攻撃を発表した．Bangerter らは 2011 年，後に Flush+Reload 攻撃と名付けられる手法を提案した[14].

これらの攻撃は time-driven, trace-driven, access-driven の三種類に分類できる[12]．time-driven attack では暗号化にかかる時間はキャッシュミスの回数と直接関係していることを利用する．キャッシュミスを推測するために攻撃者は一回の暗号化にかかる時間を利用する．trace-driven attack は一回の暗号化を行った際の，あらゆるメモリに関してのキャッシュヒットおよびミスといった詳細なデータをもとにした攻撃である．access-driven attack は書き換えられたキャッシュから，暗号化中に使用されたルックアップテーブルの入力を推測し，鍵を導出する攻撃である．Tony Arcieri はこの攻撃について，Heartbleed や POODLE などと比べ関心が低いことから CREAM と名付けた[15]．CREAM とは Cache Rules Everything Around Me の略で，OpenSSL の AES 実装においてキャッシュにより発生する時間差をもとにしたサイドチャネル攻撃の俗称である．

3.2 Bernstein によるキャッシュタイミング攻撃

2005 年，Bernstein によりキャッシュの性質を利用して AES の鍵を取得するキャッシュタイミング攻撃が提案された．AES は SubBytes, ShiftRows, MixColumns, AddRoundKey の 4 つの操作で構成されるラウンドの繰り返しにより暗号化，および復号を行う．OpenSSL を用いて AES で暗号化や復号を行う場合，プログラムとしてコードを実行する方法と，命令セットである AES-NI を用いてハードウェアとして処理する方法がある．AES-NI を用いた場合，高速に処理を行うことができ，キャッシュメモリも用いないため，キャッシュタイミング攻撃の影響は受けない．しかし，AES-NI を使用するためにはそれに対応した CPU およびソフトウェアを用いる必要がある．ここではキャッシュタイミング攻撃の対象となる，ソフトウェアによる実装について述べる．ソフトウェアによる実装では，暗号化および復号においてあらかじめ四つのルックアップテーブルを作成しておくことでラウンドの処理を高速化している．このテーブルは入力 1 バイト，出力 4 バイトで各 1024 バイトの大きさで，暗号化の処理が実行されるとき，メインメモリ上に展開される．キャッシュの機能により，一度暗号化のために参照されたルックアップテーブル内の要素はキャッシュメモリに格納される．キャッシュメモリはメインメモリと比べて容量が少ないものの，アクセス速度が高速である．CPU はルックアップテーブルを用いた暗号化を実行するとき，キャッシュメモリ内を参照し，求めている要素が存在すればメインメモリからデータを転送する代わりに，キャッシュメモリから読み出しを行う．これをキャッシュヒットという．キャッシュメモリにデータが存在せず，メインメモリから読み出しを行うことをキャッシュミスという．キャッシュミスが発生したとき，データはキャッシュメモリ内に格納され，キャッシュメモリに空きが無い場合は FIFO や LRU といったアルゴリズムによってデータの追い出しが行われる．しかしこのデータの追い出しはすべてのキャッシュラインに対して均等に行われず，システムに依存した偏りが生じる[9]．この性質を利用したのがキャッシュタイミング攻撃である．最初のラウンドでは平文$p[i]$と鍵$k^{(0)}[i]$の排他的論理和を計算したものがルックアップテーブルの入力になる．

$$x^{(0)}[i] = p[i] \oplus k^{(0)}[i], i = 0, 1, 2, \dots, 15$$

これにより攻撃者は平文と鍵が既知の状況で，ルックアップテーブルの入力に対する暗号化全体の時間を計測する．同様に攻撃者は平文が未知鍵で暗号化されたときの時間も計測する．それぞれを雑音等が無視できるほど大量のサンプルを取得し，前述の偏りとなる候補を絞り込む．偏りとなる入力に対して，既知鍵$k_k{}^{(0)}$,未知鍵$k_s{}^{(0)}$には次の関係式が成り立つ．

$$p[i] \oplus k_k{}^{(0)}[i] = p[i] \oplus k_s{}^{(0)}[i], i = 0, 1, 2, \dots, 15$$

最後に全数探索を行い，鍵を特定する．

3.3 攻撃方法

攻撃者に必要な条件は以下の通りである．

- ターゲットが，キャッシュを用いた暗号化を行っている．
- ターゲットの暗号化にかかる正確な時間を取得できる．
- ターゲットに既知鍵を用いて既知平文での暗号化を行わせることができる．
- ターゲットに未知鍵を用いて既知平文での暗号化を行わせることができる．

次に攻撃の手順について述べる．まず，ターゲットは既知鍵で大量の平文の暗号化を行い，その暗号化にかかった時間を攻撃者が計測する．次にターゲットは未知鍵で大量の平文の暗号化を行い，攻撃者は同じく暗号化にかかった時間を計測する．そして得られた入力と暗号化にかかった時間の関係を関連付けて，鍵の絞り込みを行う．最後に 絞り込んだ候補に対して全数探索を行う．

表 1　環境

	攻撃者	ターゲット
OS	Ubuntu9.10	Ubuntu9.10
CPU	AMD Athlon X2 Dual Core Processor 4200+	Intel Celeron 3.06GHz
暗号化ライブラリ	OpenSSL 0.9.8g	OpenSSL 0.9.8g
コンパイラ	GCC4.4.1	GCC4.4.1
プロセッサ数	2	1
物理コア数	1	1
コア数	2	1
L2 cache size	512KB	256KB

表 2　取得したサンプル数

	既知鍵		未知鍵	
	数	時間	数	時間
800 バイト	2^{30}	82h04m	2^{25}	2h36m
600 バイト	2^{30}	68h44m	2^{25}	2h09m
400 バイト	2^{30}	60h15m	2^{27}	7h01m

表 3　鍵候補

鍵位置	0	1	2	3	4	5	6	7
800 バイト	13 (4)	32 (3)	16 (1)	16 (1)	32 (1)	16 (5)	32 (1)	32 (1)
600 バイト	16 (2)	16 (1)	16 (1)	16 (3)	16 (1)	16 (2)	16 (1)	16 (1)
400 バイト	8 (1)	8 (1)	8 (2)	8 (2)	16 (1)	8 (3)	8 (1)	8 (2)
鍵位置	8	9	10	11	12	13	14	15
800 バイト	32 (1)	8 (7)	32 (1)	32 (2)	32 (1)	32 (2)	16 (1)	16 (4)
600 バイト	16 (2)	8 (3)	16 (1)	16 (1)	16 (3)	16 (1)	16 (3)	16 (2)
400 バイト	8 (2)	8 (1)	8 (1)	8 (1)	8 (2)	8 (1)	15 (1)	8 (1)

3.4 実装

プログラムは攻撃者とターゲットを二台の PC でそれぞれ再現する．攻撃者はまず一定の大きさのランダムパケットを生成し，ターゲットに対して送信する．これを受信したターゲットは，受信したパケットの一部を攻撃者が既知の鍵で暗号化する．この時，暗号化にかかったサイクル数も記録しておき，攻撃者に返す．攻撃者は鍵と平文を知っているので，AES の最初のラウンドでルックアップテーブルに入力される値と，それに対応するサイクル数を取得することができる．この対応を大量にパケットを送信して取得し，入力毎の平均サイクル数の統計を取る．この処理を 400, 600, 800 バイトのパケットでそれぞれ行う．次にターゲットが未知鍵で暗号化するとして，400, 600, 800 バイトのパケットで同じ統計を取る．そしてこれらのサンプルに対し，暗号化の時間が平均して他より大きい入力を見つけだし，その重みによって並び替える．最後に全数探索を行い，鍵を導出する．表 1 に実装に用いた環境を示す．

3.5 結果

取得したサンプル数を表 2 に示す．また表 3 に得られた候補数とその中での鍵の順位を示す．800 バイト，600 バイト，400 バイトについてそれぞれのサンプルの関係性により，鍵を約2^{50}の計算量で鍵を求めることができる．

4. クラウド環境におけるキャッシュタイミング攻撃の実現可能性について

4.1 クラウド環境での Bernstein の手法の検証

三章では Bernstein の手法の有効性について確認したが，現状では以下に挙げる三つの欠点により，現実的な脅威としては認識されていない．

- 攻撃を行うために，予め既知鍵でのサンプルを収集する必要があること
- 暗号化にかかる時間を正確に計測する必要があること
- サンプル数が大量に必要であること

Bernstein による手法では暗号化にかかった正確な時間を計測するために，攻撃者がサーバの PC のサイクル数を取得できる，という仮定でシミュレーションを行っている．これについて論文中では，サンプル数を増やすことでノイズを相殺できる，と述べられているが，実際に検証されていない．この時に発生するサイクル数の差はおよそ 3~10 サイクルである．実際の通信環境を考えると，このような時間差を計測するのは極めて困難であると考えられる．

そこで，このサイクル数を別の方法で攻撃者が取得する方法としてクラウド環境を想定する．クラウド環境は一台の PC に複数の仮想マシン(VM)を動作させることで提供されるサービスで，ユーザが期間やスペックを必要に応じて VM を使用する．仮想化技術には主にホスト OS 型とハイパーバイザ型が存在するが，ここではクラウドで多く用いられるハイパーバイザ型について述べる．ハイパーバイザ型の仮想化ではハードウェアの BIOS から直接仮想化ソフトを起動して，その上で仮想マシンを実行する．このためホスト OS 上で仮想化ソフトを実行するホスト OS 型に比べ，オーバーヘッドが小さくパフォーマンス面で有利である．VM はそれぞれ仮想 CPU を保持しており，仮想 CPU は VM から見ると OS やアプリケーションを動作させてい

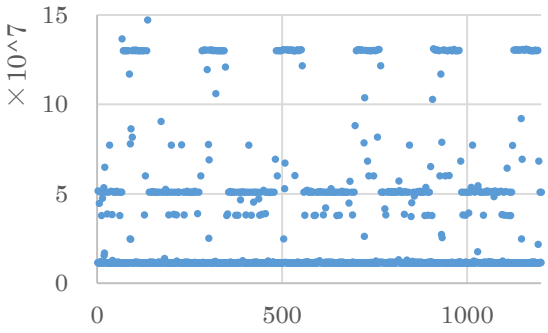

図 1　サイクル数の変化

る実際の CPU のように見える．しかし実際はハイパーバイザが仮想 CPU に CPU リソースを割り当てている．ここで，各 VM に割り当てられた仮想 CPU のコア数が実際の CPU のコア数よりも多いとすると，VM 間で CPU コアの競合が発生する．この時，お互いに動いているプロセスに要するサイクル数が影響しあうことで，他の VM のサイクル数を取得できないかを考える．

ホストとして Ubuntu14.04 LTS, Intel Core i3-2120 の PC 上に KVM によるハイパーバイザ型の仮想環境を作り，そこにゲスト OS として Ubuntu 9.10 が入った VM を二台用意した．各 VM には仮想 CPU を一つ割り当て，ハイパーバイザ側で VM が使用する CPU コアを同一のものに割り当てて競合を発生させた．この時，一台の VM で一定の間隔で負荷をかけるプロセスを実行し，もう一台で一定処理のプロセスを実行することで，サイクル数が計測できないかを考える．図 1 に一定処理を行ったときのサイクル数の変化を示す．このグラフから，一定処理を繰り返したにもかかわらず，VM 内で要したサイクル数は大きく変化していることがわかる．これは，他方の VM が CPU 資源を大量に使用しているため，一定の処理であっても同じサイクル数で処理が完了しないからであると考えられる．

4.2 キャッシュをクリアする手法

VM 間でサイクル数が相互作用する性質を利用して，新たな攻撃が出来ないかを考える．Bernstein による手法では暗号化に要した CPU サイクルの取得が必要であった．この CPU サイクルの変化を生み出しているキャッシュミスは攻撃者側で操作することができないため，大量のサンプル数が必要であった．ここで，攻撃者が能動的にキャッシュを操作して暗号化にかかる速度を観測する攻撃を考える．VM がコアを共有している環境ではキャッシュも共有しているため，攻撃者がキャッシュをクリアすればもう一台の VM 上で行われている暗号化の速度を間接的に操作することができると考えられる．この時の速度の違いを計測することで，鍵回復を行えないか今後検討を行う．

5. むすび

本稿では SSL / TLS に対する鍵回復攻撃として提唱されている Bernstein の キャッシュタイミング攻撃およびその改良方式が，現実的な環境，例えばクラウド環境において適用可能であるか検証し，考察を与えた．実際に鍵回復を行うためには攻撃対象側の CPU クロック消費数を詳細に観測する必要がある．しかしながら，必ずしもそのクロック数を計数する必要はなく，暗号化速度とキャッシュヒット数の微妙な差が観測できることで推定可能であることも示した．今後，攻撃側がキャッシュを共有し，かつ制御できる環境下で，十分な量の暗号化処理が観測できるとした場合の鍵回復可能性について検討を行う予定である．

参考文献

[1] A. Freier, P. Karlton, Netscape Communications, P. Kocher, “The Secure Sockets Layer (SSL) Protocol Version 3.0”, https://tools.ietf.org/html/rfc6101, 2015 年 6 月 24 日参照.

[2] T. Dierks, C. Allen, “The TLS Protocol Version 1.0”, https://www.ietf.org/rfc/rfc2246.txt, 2015 年 6 月 24 日参照.

[3] Daniel J. Bernstein “キャッシュタイミング攻撃 on AES”, http://cr.yp.to/antiforgery/cachetiming-20050414.pdf, 2015 年 6 月 24 日参照.

[4] Thai Duong, Juliano Rizzo, “Here Come The Ninjas”, http://www.hpcc.ecs.soton.ac.uk/~dan/talks/bullrun/Beast.pdf, 2015 年 6 月 24 日参照.

[5] Juliano Rizzo, Thai Duong “The CRIME attack” https://docs.google.com/presentation/d/11eBmGiHbYcHR9gL5nDyZChu_-lCa2GizeuOfaLU2HOU/edit, 2015 年 6 月 24 日参照.

[6] Angelo Prado, Neal Harris, Yoel Gluck, “SSL, GONE IN 30 SECONDS A BREACH beyond CRIME” https://media.blackhat.com/us-13/US-13-Prado-SSL-Gone-in-30-seconds-A-BREACH-beyond-CRIME-Slides.pdf, 2015 年 6 月 24 日参照.

[7] Nadhem J. AlFardan, Kenneth G. Paterson, “Lucky Thirteen: Breaking the TLS and DTLS Record Protocols”, http://www.isg.rhul.ac.uk/tls/TLStiming.pdf, 2015 年 6 月 24 日参照.

[8] Bodo M\"oller, Thai Duong, Krzysztof Kotowicz, “This POODLE Bites: Exploiting The SSL 3.0 Fallback”, https://www.openssl.org/~bodo/ssl-poodle.pdf, 2015 年 6 月 24 日参照.

[9] Michael Neve and Jean-Pierre Seifert and Zhenghong Wang, “Cache time-behavior analysis on AES”, http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.4091&rep=rep1&type=pdf, 2015 年 6 月 24 日参照

[10] Hassan Aly and Mohamed Elgayyar “Attacking AES Using Bernstein's Attack on Modern Processors”. http://www-users.math.umn.edu/~math-sa-sara0050/teaching/14s/AlyElgayyar.pdf, 2015 年 6 月 24 日参照.

[11] Trustworthy Internet Movement, “SSL Pulse”, https://www.trustworthyinternet.org/ssl-pulse/, 2015 年 6 月 27 日参照

[12] Hassan Aly and Mohamed Elgayyar, “Attacking AES Using Bernstein's Attack on Modern Processors”, http://www-users.math.umn.edu/~math-sa-sara0050/teaching/14s/AlyElgayyar.pdf, 2015 年 6 月 27 日参照

[13] John Kelsey, Bruce Schneier, David Wagner, Chris Hall, “Side Channel Cryptanalysis of Product Ciphers”, https://www.schneier.com/paper-side-channel2.pdf, 2015 年 6 月 27 日参照

[14] Endre Bangerter, David Gullasch, Stephan Krenn, “Cache Games – Bringing Access-Based Cache Attacks on AES to Practice”, https://eprint.iacr.org/2010/594.pdf, 2015 年 6 月 27 日参照

[15] Tony Arcieri “CREAM: the scary SSL attack you’ve probably never heard of”, http://tonyarcieri.com/cream-the-scary-ssl-attack-youve-probably-never-heard-of, 2015 年 6 月 27 日参照