RAMBleedによるOpenSSLの秘密情報の回復

冨田 千尋[1,a)]　瀧田 愼[2]　福島和英[3]　仲野有登[3]　白石 善明[1]　森井 昌克[1]

概要：DRAM（Dynamic Random Access Memory）の一つのアドレスにアクセスを繰り返して，その近隣でビット反転を誘発することを Rowhammer という．RAMBleed は Rowhammer を利用したサイドチャネル攻撃であり，一般ユーザのアクセス権限がない秘密情報を読み取り可能である．本論文では，OpenSSL により TLS 通信が実装された Apache Web サーバに対して，RAMBleed を用いてサーバの秘密情報の回復できることを明らかにする．まず，OpenSSL および Apache の挙動を解析し，TLS ハンドシェイクを実行する際に，RSA の秘密鍵生成に用いられる 2 つの素数がメモリ上の特定の位置に展開されることがわかった．これらの秘密情報の RAMBleed による読み取り結果には一部に誤りが含まれるが，RSA 暗号の解析手法を用いることで誤りのない秘密情報を回復可能である．我々は，OpenSSL で用いられる RSA 秘密鍵を回復するまでの RAMBleed を含む一連の攻撃を実装し，高い確率で秘密情報を取得できることを示した．

キーワード：rowhammer, rambleed, openssl, 秘密鍵, サイドチャネル攻撃

Extracting Secrets of OpenSSL with RAMBleed

Chihiro Tomita[1,a)]　Makoto Takita[2]　Kazuhide Fukushima[3]　Yuto Nakano[3]
Yoshiaki Shiraishi[1]　Masakatu Morii[1]

Abstract: There exists an attack called Rowhammer, in which repeated accesses to rows of DRAM induce bit flipping within its neighboring rows. A side-channel attack called RAMBleed, which is based on the Rowhammer attack, can extract secrets that the attacker cannot access, and has been reported to be applicable to OpenSSH. In this paper, we apply RAMBleed to the Apache Web server, which implements TLS using OpenSSL. As a result of analyzing the behavior of OpenSSL and Apache, we found that when a newly created child process executes the TLS handshake, the two prime numbers used to generate the RSA secret key are placed in memory and are located at the same offset in the memory page. Using this behavior, we show that an attacker can force the server to spawn a child process to induce the secrets in an arbitrary memory location and recover the secrets with high probability.

Keywords: Rowhammer, RAMBleed, openssl, secret key, side-channel attack

1. はじめに

計算機やスマートフォンにはDRAM（Dynamic Random Access Memory）と呼ばれる半導体メモリが使用されている．半導体の微細化，高密度化が進むとともに，大容量のDRAMが作られるようになり計算機の性能が向上している．その一方で，高密度化により DRAM 内のビット同士の物理的距離が近づいた結果，あるビットへのアクセスが別のビットに影響を与え，意図しないビット反転が生じることが指摘されている．この現象は Rowhammer[1] と呼ばれ，それを悪用した権限昇格攻撃 [2]，故障利用攻撃 [3]，DoS 攻撃 [4] などが提案されている．

Kwong らは 2019 年に Rowhammer に基づく攻撃とし

[1] 神戸大学大学院工学研究科
Graduate School of Engineering, Kobe University
[2] 兵庫県立大学大学院情報科学研究科
Graduate School of Information Science, University of Hyogo
[3] 株式会社 KDDI 総合研究所 情報セキュリティグループ
KDDI Research, Inc. Information Security Laboratory
[a)] tomita@stu.kobe-u.ac.jp

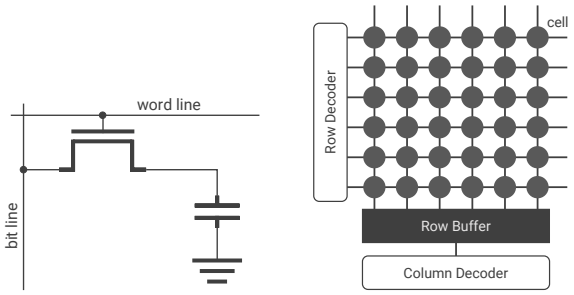
て，RAMBleed と呼ばれるサイドチャネル攻撃を提案した [5]．RAMBleed は，意図的に引き起こした Rowhammer によるビット反転を観察することにより，アクセス権限のない秘密情報の一部を高い精度で回復する攻撃である．これにより，OpenSSH で利用される秘密鍵の情報を取得できることも同時に示されている．

RAMBleed を実行するためには，攻撃者のプロセスの中で攻撃対象のプロセスを制御し，回復したい情報を DRAM 上の意図したアドレスに誘導する必要がある．Kwong らは Linux のメモリ割り当てアルゴリズムである Buddy Allocator を利用して秘密情報を誘導を実現できることを示している．実際に，特定の秘密情報を誘導するためには，攻撃対象のプロセスの中でどのような順番でどの情報にメモリが割り当てられるかを解析する必要がある．OpenSSH の場合は，TCP 接続を受けた SSH デーモンによって生成される子プロセスがクライアントを認証するときに，RSA 秘密鍵に対して物理メモリが新たに割り当てられる．秘密情報の前に割り当てられる情報に必要なメモリの量が分かれば，Buddy Allocator によりメモリ割り当てをコントロールし，秘密情報を所望のアドレスに配置できる．したがって，RAMBleed を実行するためには，秘密情報に新たに物理メモリが割り当てられることと，その割り当てのタイミングが高確率で予測可能であることが求められる．

本論文では，OpenSSL により SSL/TLS 通信に対応した Apache Web サーバで利用される RSA 秘密鍵の情報の読み取りを試みる．本論文の主な貢献は，未だ回復手法が提案されていない OpenSSL で利用される秘密情報を，管理者権限を必要としない RAMBleed により読み取り可能であることを明らかにしたことである．

具体的には，サーバが保有する RSA 秘密鍵を構成する情報を Buddy Allocater により誘導できるかどうかを調査した．マルチプロセッシングモデルとして prefork モデルを採用している Web サーバの動作を解析した結果，新たに生成された子プロセスが TLS ハンドシェイクを実行するとき，RSA 秘密鍵の生成に用いられる 2 つの素数に物理メモリページが割り当てられることを明らかにした．また，prefork モデルの挙動を利用し，子プロセスを強制的に生成させることで，高い確率で新規子プロセスと TLS ハンドシェイクを実行できることを示した．さらに，素数に物理メモリが割り当てられるタイミングを繰り返し観測することで，そのタイミングに偏りがあることがわかった．これらの結果をもとに，RSA 秘密鍵の生成に用いられる二つの素数を，RAMBleed で情報を読み取るために用意した DRAM 上の特定のアドレスに高い確率で誘導できることを実験により示した．

また，我々は前述の結果を利用して実際に RAMBleed で素数を読み取る実験を行った．その結果，秘密情報の一部を約 95%と，高い精度で読み取ることに成功している．



(a) セル　(b) Bank

図 1　DRAM の構造

RAMBleed で読み取れる素数の情報は一部であり，数%の誤りが含まれているものの，RSA 暗号に対する攻撃手法を組み合わせて利用することで，誤りのない素数を復元できる．まず，誤りが含まれる素数のビット列を正しいものに復元するアルゴリズムとして，Paterson らのアルゴリズム [9] がある．このアルゴリズムでは，RSA 秘密鍵を構成する 2 つの素数が満たす関係式を利用して，最下位ビットから順に復元できる．RAMBleed による読み取りで生じた数%の誤りは，Paterson らのアルゴリズムで復元できることを実験的に確認している．加えて，Coppersmith のアルゴリズム [12] によって，復元した素数の部分情報から全てのビット値を復元できる．以上の結果をまとめると，RAMBleed を含む一連の攻撃で OpenSSL により実装された SSL/TLS 通信で用いられる RSA 秘密鍵をメモリ上から読み取り，復元できることが明らかとなった．

2. RAMBleed

2.1 DRAM

DRAM(Dynamic Random Access Memory) は，図 1(a) のようにキャパシタとトランジスタから構成されるセルで 1 ビットの情報を保持する．充電された状態で 1，充電されていない状態で 0 を表すセルを true セルと呼ぶ．逆に，充電された状態で 0，充電されていない状態で 1 を表すセルを anti セルと呼ぶ．本論文では，簡単のために true セルのみを考慮して説明する．

セルは図 1(b) のような行列の形に並べて配置されており，各行はワード線で，各列はビット線で接続されている．一定数の行をまとめて Bank と呼び，複数の Bank から DRAM チップが構成される．さらに，複数の DRAM チップをまとめて Rank と呼び，メモリモジュールである DIMM の片面に実装される．

DRAM 内でのデータの扱いは，8KiB から成る行単位でなされる．次のようにしてデータへアクセスする．目的のデータが含まれる行のワード線の電圧を高くすることで，キャパシタがビット線に接続される（行をアクティブにする）．これによって，1 行分のデータが図 1(b) に示す行バッファへ転送される．その後，CPU は行バッファ内にあるデータから目的の列にアクセスして，データの読み取

り/書き込みを行う．

DRAM セルのキャパシタに貯められている電荷は，時間の経過とともに徐々に失われていく．これによるデータの損失を防ぐために定期的にキャパシタを充電し直す必要があり，この動作をリフレッシュをいう．

2.2 Linux Buddy Allocator

Linux では，システムに搭載されているメモリを，ページと呼ばれる領域（一般的には 4KiB）を最小単位として管理する．buddy allocator というアルゴリズムを使い，このページの割り当てや解放を行う．カーネルは，メモリの全ての空き領域を，物理的に連続するページごとに一つのブロックとしてオーダー 0 からオーダー 10 に分類する．オーダー n は 2^n ページのブロックで構成され，各オーダーでメモリブロックは，スタックのような first-in-last-out(FILO) のデータ構造をしている．

プロセスからメモリ割り当ての要求があった場合, Buddy Allocator は必要な領域に対して最小のブロックを割り当てる．ただし，ユーザ空間からはオーダー 0 の要求のみが可能となっている．ユーザ空間からメモリ割り当てを要求されると，サイズに関わらずオーダー 0 にあるブロックから順に割り当てる．

2.3 Rowhammer

Rowhammer は DRAM の複数の行にアクセスを繰り返すことによって，その近隣の行でビット反転が誘発されるという現象である [1]. 反転を起こしたいメモリ領域に直接アクセスすることなくデータを書き換えることが可能となるため，コンピュータシステムのセキュリティにおける重大な問題であると考えられている．

セル密度の増加に伴い，ワード線が互いに近接して配置された結果，あるワード線の電圧が上げられることで近隣のワード線の電圧が部分的に高くなり，その位置にあるセルから電荷が失われる．短時間に同じ行を何度もアクティブにすることでこの影響は大きくなり，リフレッシュが行われる前にノイズマージンを超えるだけの電荷が失われると，ビット反転が起こる．

Rowhammer によるビット反転の起こりやすさは，上下にあるビットの値に依存する．具体的には，充電されたセルの上下に充電されていないセルがある場合に，ビット反転は最も起こりやすくなる [1]．簡単な例を以下に示す．同じ列にある隣接した 3 つのビットの値を上から順に $x-y-z$ と表すこととする．true セルの場合，ビットの値が 1 のセルは充電されている．そのため，$0-1-0$ の状態が最もビット反転が起こりやすい．この状態で上下のビットを含む行に繰り返しアクセスすることで，真ん中のビットで反転が起こり $0-0-0$ に変化する可能性がある．また，この現象は $0-1-1$ の場合にも発生し得る．

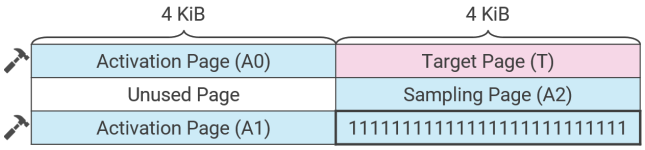


図 2 single-sided RAMBleed で用いるレイアウト

2.4 RAMBleed

Kwong らは Rowhammer を利用して，データの機密性に影響を与える RAMBleed を提案した [5]．RAMBleed は，攻撃者が Rowhammer によるビット反転を利用して，その上下にあるビットの値を高い確率で推測する攻撃である．これにより，攻撃者は直接アクセスできないメモリ領域に格納されている秘密鍵などといった情報を読み取ることが可能となる．攻撃にはメモリの割り当てと解放，攻撃プロセスが所有するメモリ領域へのアクセスができれば十分であるため，攻撃者は特権を必要としない．

また，我々は，従来の RAMBleed と比較して秘密情報の誘導回数を半数に削減しつつ，読み取り精度を保つ新たな single-sided RAMBleed を提案した [6]．以下では，この single-sided RAMBleed について主要な部分の説明を行う．

RAMBleed では，次の 2.4.1 項から 2.4.3 項の手順で情報を読み取る．

2.4.1 メモリレイアウトの作成

攻撃者はまず，物理メモリ上の隣接した 3 行を確保して，図 2 に示すレイアウトを作成する．右下のページ内の全てのビット値は 1 に設定する．それぞれのページサイズは 4KiB であり，各行に 2 ページが含まれる．ここで，図中の A0，A1，A2 で示した部分は攻撃者のプロセスに割り当てるページであり，T で示した部分は攻撃対象となるプロセスに割り当てるページである．

レイアウトを作成した後，攻撃者は Rowhammer 攻撃を実行する．ここで，DRAM 内での行バッファへの転送は行単位でまとめて行われる．例えば，攻撃者が A0 のページにアクセスすると A0 と T の 2 ページ分のデータが行バッファへ転送されることとなる．そのため，A0 と A1 の 2 ページに何度も繰り返しアクセスして Rowhammer 攻撃を行うことで，A2 内でのビット反転を誘発することができる．また，A2 は攻撃者のプロセスに割り当てるページのため，攻撃者はデータの読み書きが可能であり，ビット反転が起こったかどうかを確認できる．

2.4.2 秘密情報の配置

攻撃者は秘密情報を読み取るために，攻撃対象となるプロセスを起動し，Target Page の位置に読み取りたい情報を含むページを誘導する必要がある．そこで，Kwong によって提案された，buddy allocator の動作を利用した FFS（Frame Feng Shui）と呼ばれる手法 [5] を用いる．この方法では，攻撃対象のプロセスが秘密情報にアクセスするまでに割り当てを受ける物理メモリページ数 (Dummy Page

図 3 秘密情報を配置中の Buddy Allocator の様子

数) を，攻撃者が知っていることが前提となる．

step1．ページの確保

攻撃対象のプロセスが起動後，$n+1$ ページ目に目的の秘密情報を格納すると仮定する．このとき，攻撃者は n ページ分の領域を確保しておく．

step2．ページの解放

攻撃者は，秘密情報を置かせたいページ（Target Page）を解放する．その後，step 1 で割り当てられた n ページを全て解放する．この結果，図 3 のように，Buddy Allocator のオーダー 0 の最上位に n ページ分の領域があり，その直後に Target Page がある状態になる．

step3．攻撃対象プロセスの起動

step 2 の直後に，攻撃者は攻撃対象のプロセスを起動することで，オーダー 0 の初めの n ページが割り当てられた後，Target Page に秘密情報が格納される．

2.4.3 情報の読み取り

RAMBleed では，ビット反転のデータ依存性を利用した秘密情報の読み取りを行う．まず，あるページ P の先頭から i 番目（$i \in \{0, 1, ..., 32767\}$）のビットを $P[i]$ で表す．A2 に反転が起こるビットが含まれていると，Target Page に格納されているデータの一部を次のようにして読み取ることができる．攻撃者はあらかじめ A2 内でビット反転が起こる位置を特定し，それを保持しておく．ここでは，$A2[i]$ でビット反転が見つかったとする．次に，$A2[i]$ の値を 1 にする．その後，A0 と A1 へ繰り返しアクセスし，A2 に対して Rowhammer 攻撃を行う．最後に，攻撃者は $A2[i]$ の値を確認する．

$T0[i]$ と $T1[i]$ の値が共に 1 の場合，この 1 列は $1-1-1$ となるため，$A2[i]$ でのビット反転が起こりにくい．逆に，$T0[i]$ と $T1[i]$ の値が共に 0 の場合，この 1 列は $0-1-1$ となるため，$A2[i]$ でビット反転が起こり $0-0-1$ となりやすい．よって，Rowhammer 攻撃を行った後は高い確率で $T0[i], T1[i]$ の値と $A2[i]$ の値は一致する．したがって，$A2[i]$ の値を確かめることで攻撃者は情報の一部を高い確率で読み取ることができる．

3. OpenSSL と Apache について

3.1 SSL/TLS

SSL/TLS とは，PKI(Public Key Infrastructure) を技術基盤として，セキュアな通信を実現するプロトコルである．共通鍵暗号方式による通信内容の暗号化や，公開鍵暗号方式による認証によって，盗聴や改ざん，なりすましを防ぐ．認証時に用いられる公開鍵暗号方式として，RSA 暗号がある．RSA 暗号を用いた認証では，公開鍵とサーバのドメインを紐づけたサーバ証明書により，クライアントが通信しようとしている相手が，意図したドメインの秘密鍵を所有していることを確認できる．

サーバ証明書は，サーバ管理者が秘密鍵を用いて作成する CSR(Certificate Signing Request) を元に，認証局によって発行される．したがって，秘密鍵が漏洩すれば証明書の第三者による複製が可能となる．また，TLS1.2 以前のバージョンでは，暗号化通信に用いる共通鍵の交換に RSA 暗号が利用される場合がある．RSA 暗号による鍵交換では，共通鍵の元となる情報がサーバが保有する秘密鍵に対応する公開鍵で暗号化されてやりとりされる．そのため，漏洩した秘密鍵を持つ第三者がこのやりとりを傍受し，共通鍵を複製する恐れがある．このように，秘密鍵の漏洩は，SSL/TLS プロトコルの安全性が損なわれる重大な問題である．

SSL/TLS 通信対応の Web サーバを構築する場合，オープンソースで開発されている OpenSSL[7] がよく用いられる．SSL/TLS プロトコルの最新のバージョンである TLS1.3 の機能も実装されている．

3.2 Apache

Apache[8] はオープンソースで開発されている Web サーバソフトウェアで，世界中で利用されている．OpenSSL と組み合わせることで，HTTPS 通信に対応した Web サーバを構築できる．

Apache では，複数のクライアントからのリクエストに同時に対応するために 3 種類のマルチプロセッシングモデルが実装されており，その一つとしてリクエストの数に応じて動的にプロセス数を調節する prefork モデルがある．prefork モデルでは，サーバの親プロセスが事前に一定数の子プロセスを生成し，リクエストを処理していないアイドル状態の子プロセスにリクエストに振り分けて対応させる．親プロセスがアイドルプロセス数以上のリクエストを受信した場合には，新たに子プロセスを生成して対応する．また，prefork モデルでは各子プロセスに対してメモリが割り当てられるため，メモリの消費を抑制するためにアイドルプロセス数を制限する必要がある．Apache では最大，最小アイドルプロセス数がそれぞれ MaxSpareServers, MinSpareServers というパラメータで定められている．アイドルプロセス数が MaxSpareServers の値を上回れば親プロセスがアイドル状態の余分な子プロセスを終了させ，MinSpareServers の値を下回れば，新たな子プロセスが生成される．

4. RSA 秘密鍵を復元するアルゴリズム

RAMBleed で読みった RSA 秘密鍵の各ビットの値は必ずしも全て正しい値ではなく，誤りを含んでいる．ノイズのある秘密鍵から正しい秘密鍵を復元するアルゴリズムとして Heninger らのアルゴリズム [10]，Henecka らのアルゴリズム [11]，Paterson らのアルゴリズム [9] などが存在する．ここでは，RAMBleed で得られる秘密情報に適した Paterson らのアルゴリズムを紹介する．以下，ノイズのある秘密鍵における 0 から 1，1 から 0 へのビット反転率を，それぞれ α，β とする．

本論文では，秘密鍵を構成する 2 つの素数 p, q を RAMBleed で読み取る対象とするため，p, q を復元する Paterson らのアルゴリズムについて述べる．Paterson らのアルゴリズムでは p, q が最下位ビットから t ビットずつ復元される．ここで，t ビットを復元する手順の概要を示す．

step1. t ビットのビット列の全パターンを，拡張する p の候補とする．各候補に対して，対応する q の値が，公開鍵と p, q が満たすべき関係式により定まる．したがって，2^t 個の (p, q) のペアが候補となる．これにより 2^t 個の候補が生まれる．

step2. 各候補に対して，ノイズのあるビット列を用いて尤度を計算する．尤度は，各候補を読み取ったときに，ノイズのあるビット列が読み取られる確率（事後確率）で定められる．

step3. 尤度の高い順に L 個の候補を残す．残った L 個の候補に対して step1 を実行する．

そして，step3 で残った L 個の候補に対して step1 を実行する．この操作を，必要なビット数が得られるまで繰り返す．α,β が大きいほど，大きな t と L が必要となり，復元精度が低下する．

また，Coppersmith らは，p，もしくは q の最下位ビットから $1/4 \log_2 N$ ビットが得られたとき，各素数の全ビットを復元できることを示している [12]．

5. OpenSSL，Apache の解析

本論文では，サーバが保有する RSA 秘密鍵を構成する情報を RAMBleed で読み取る対象とする．RAMBleed を成功させるためには，2.4.2 で示した方法で秘密情報を誘導する必要がある．この方法は，攻撃者が Dummy Page 数を把握していることが前提となっている．

本章では，OpenSSL により SSL/TLS 通信に対応した Apache Web サーバの挙動を解析し，Dummy Page 数を事前に導出する方法を与える．また，実験結果により，実際に Dummy Page 数を高い確率で予測できることを示す．

表 1 調査・実験環境

OS		Ubuntu 18.04
CPU		Core i5-4590
DRAM	規格	DDR3
	サイズ	4GiB × 2 枚
	周波数	1333MHz
	ECC	非対応
Apache		2.4.46
OpenSSL		1.1.1

5.1 調査・実験環境

調査・実験環境を表 1 に示す．自身で作成した秘密鍵を用いて自己証明書を発行し，サーバの設定ファイルに登録することで Apache Web サーバを TLS 通信に対応させた．また，サーバのマルチプロセッシングモデルを prefork モデルとした．その他の設定はデフォルトである．

5.2 秘密情報への物理メモリ割り当ての有無

本節では，Web サーバのプロセスが起動したとき，すなわちクライアントからの TLS リクエストを受信したときの秘密情報への物理メモリページの割り当ての有無を調査する方法と，その結果を述べる．対象とする秘密情報は，RSA 秘密鍵の生成に用いられる 2 つの 1024 ビットの素数 p, q とする．

5.2.1 秘密情報の仮想アドレスの特定

prefork モデルでは，fork() システムコールによって子プロセスを生成している．fork の実行直後は，COW により親プロセスと子プロセスが同一の仮想メモリを共有することで，不要なメモリ消費を抑制する．しかし，子プロセスが仮想メモリページに書き込み処理を実行したとき，共有している仮想メモリ内の該当ページが複製され，そのページには子プロセスのみがアクセスできるようになる．このとき，複製された仮想メモリページには物理メモリページが割り当てられていないため，ページフォルトが発生し，カーネルが新たに物理メモリページを割り当てる．

我々はこの動作に着目し，子プロセスが TLS ハンドシェイクを実行する際に秘密情報を仮想メモリに書き込み，物理メモリページの割り当てが発生するという仮定の下で簡便な調査を行った．調査を簡単にするために MaxSpareServers の値を 1 としてアイドル状態の子プロセスが常に 1 つとなるように設定し，通信を行う子プロセスを限定した．また，TLS ハンドシェイク実行時には SSL/TLS バージョンを TLS1.3, 暗号スイートを AES256-GCM-SHA384 とクライアントから指定した．

簡便な調査は以下の手順で行う．

step1. gcore コマンドを用いてアイドル状態の子プロセスの仮想メモリダンプ A を取得する．

step2. TLS リクエストを送信し，1 つのみ存在する子プ

ロセスと TLS ハンドシェイクを実行した後，TLS 通信を行っている子プロセスの仮想メモリダンプ B を取得する．

step3. 仮想メモリダンプ A,B を探索し，秘密情報の仮想アドレスを特定する．B のみに存在する秘密情報の仮想アドレスがあれば，そのアドレスを持つ秘密情報は TLS ハンドシェイクの実行時に子プロセスの仮想メモリに書き込まれたことが分かる．

以上の操作を，生成されてから TLS ハンドシェイクを実行したことのある子プロセスと，そうでない子プロセスに対して行った．その結果，既に TLS ハンドシェイクを実行した子プロセスでは，TLS ハンドシェイクの実行前後で p, q の仮想アドレスに変化はなく，新しい秘密情報も書き込まれなかった．一方で，生成されてから TLS ハンドシェイクを実行したことのない子プロセスにおいては，TLS ハンドシェイク実行後に仮想メモリ上に新たに素数 p, q が書き込まれることが分かり，それらの仮想アドレスを特定できた．

5.2.2 p, q への物理メモリ割り当て

次に，仮想メモリへの書き込みを確認した p, q を含むページの物理アドレスが，TLS ハンドシェイクの実行前後で変化するかを調べる．物理アドレスの変化は，p, q に新たに物理メモリが割り当てられたことを示す．物理メモリが割り当てられていない仮想メモリページの物理アドレスは 0 とする．

ある情報に割り当てられた物理アドレスを確認するためには，サーバプロセスの各仮想メモリページの仮想アドレスと，それに対応する物理アドレスを格納するページテーブルを参照する必要がある．CONFIG_PROC_MONITOR=y としてビルドされた Linux カーネルでは，プロセスのページテーブルに/proc/(対象プロセスのプロセス ID)/pagemap から管理者権限でアクセスできる．

調査の結果，TLS ハンドシェイクの実行前では p, q に物理メモリが割り当てられておらず，実行後に新たに割り当てられることが分かった．

5.2.1 の結果と合わせると，子プロセスが初めて TLS ハンドシェイクを実行するときに，p, q に物理メモリが割り当てられてることが確認できた．

5.2.3 新規子プロセスとの TLS ハンドシェイク実行方法

前項で述べた p, q への物理メモリページの割り当てを実際の通信プロセスにおいて発生させるには，攻撃者がサーバに子プロセスを生成させ，かつ新しく生成された子プロセスのいずれかと TLS ハンドシェイクを実行する必要がある．本節では，prefork モデルの動作を利用してこれを実現する手法について述べる．手順は以下のとおりである．

step1．MaxSpareServers 個の TCP リクエストを送信

サーバには最大で MaxSpareServers 個のアイドルプロセスが存在する．そこで，MaxSpareServers 個の TCP リクエストを送信すると，サーバは既存のアイドル状態の子プロセスにリクエストを割り当て，子プロセスが足りない場合は MinSpareServers 個のプロセスがアイドル状態となるまで子プロセスを生成する．これにより，少なくとも MinSpareServers 個の新規子プロセスを生成させることができる．このような新規子プロセスを生成させるためのクライアントをダミークライアントと呼ぶ．

step2．TLS リクエストを送信

Apache では，子プロセスがクライアントから TCP パケットを受信してから次の TCP パケットを受信するまでの待機時間が Timeout というパラメータで定められており，待機中の子プロセスは非アイドル状態となる．ダミークライアントから FIN パケットを送信しない限り，既存の子プロセスはアイドル状態とはならないため，Timeout で設定されている時間はサーバに存在するアイドルプロセスは新規子プロセスのみとなる．この状態で攻撃者が Timeout で設定されている時間内に TLS リクエストを送信すると，高い確率で新規子プロセスと TLS ハンドシェイクを実行できる．なお，親プロセスが複数の子プロセスを生成・終了させるとき，1 秒間に 1 つのプロセスを処理するため，ダミークライアントからのリクエスト送信後に 1 秒以上待機してから TLS リクエスト送信する必要がある．

この方法を用いれば，攻撃プロセスは高い確率で新規に生成された子プロセスと TLS ハンドシェイクを実行でき，p, q への物理メモリ割り当てを発生させることが可能である．

5.3 Dummy Page 数の測定

5.2 の方法で p, q に物理メモリを新規に割り当てることができる．次に，p, q の誘導に必要な Dummy Page 数を実験により求める．以降の実験では，MaxSpareServers の値をデフォルト値である 10 とした．

5.3.1 測定方法

Dummy Page 数は，対象となる秘密情報を保有するプロセスのプロセス ID と，そのプロセスの仮想アドレス空間における秘密情報の仮想アドレスが既知である場合に，以下の 5 ステップで測定できる．

step1. 物理メモリページを 100 ページ獲得し，各ページの物理アドレスを記録する．

step2. step1 で獲得したページを 1 ページずつ全て解放し，各ページの BuddyAllocator における order0 での並び順を記録する. 解放したページは order0 上でスタックのデータ構造で管理されるため，最後に解放したページを先頭とした順番を記録する．

step3. サーバに MaxSpareServers 個のリクエストを送

表 2 p の Dummy Page 数測定結果

Dummy Page 数	測定された回数
35	84
36	16

表 3 q の Dummy Page 数測定結果

Dummy Page 数	測定された回数
42	81
43	17
53	1
90	1

信し，その 1 秒後に TLS リクエストを送信する．

step4. 測定プロセスが通信しているサーバプロセスのプロセス id と秘密情報の仮想アドレスから秘密情報が含まれるページの物理アドレスを求める．

step5. step2 で解放したページのうち，物理アドレスが秘密情報の物理アドレスと一致するページの order0 上での並び順から 1 を引いた数を Dummy Page 数とする．

以上の 5 ステップによる Dummy Page 数の測定を繰り返し，Dummy Page 数の偏りを観測する．

なお，COW により，異なる子プロセス間で p, q の仮想アドレスは一致する．p, q の書き込みによって複製された仮想メモリページの仮想アドレスは親プロセスと共有していた仮想メモリ内でのアドレスとなる．したがって，各子プロセスが TLS ハンドシェイクにおいて同じ仮想メモリページに p, q を書き込む場合，子プロセス間でそれらの仮想アドレスは変化しない．

5.3.2 Dummy page 数の測定

まず，素数 p を対象として，上記の 5 ステップを実行する測定プログラムを 100 回実行した結果を表 2 に示す．測定の結果，Dummy Page 数に大きな偏りが見られ，33 ページが最も多く，83%の割合で計測された．

次に，素数 q を対象として測定した結果を表 3 に示す．q についても p と同様に測定されたページ数に偏りが見られ，42 ページが 81%の割合で計測された．

2 つの値が測定されている原因としては，測定プロセスから解放した 100 ページのうちのいずれかページが，TLS ハンドシェイクを実行しているプロセス以外のプロセスに割り当てられたことが原因と考えられる．

素数 p, q ともに Dummy Page 数に大きな偏りが見られたので，これらの値をもとに Dummy Page 数を設定すれば，2.4.2 の方法で p または q を誘導でき，RAMBleed によりその部分情報を読み取り可能である．

6. OpenSSL の秘密情報の読み取り

本章では, RAMBleed を含む一連の攻撃により, OpenSSL で実装された SSL/TLS 通信に利用される RSA 秘密鍵を読み取り，復元する手法と，その実験結果について述べる．

表 4 RAMBleed による p の読み取り結果

Correct	95.6%
$P(0 \to 1)$	5.7%
$P(1 \to 0)$	3.1%

攻撃の手順を以下に示す．

step1. 以下の 3 ステップを必要なビット数が得られるまで繰り返し，メモリ上の秘密情報を読み取る．

step1-1. Sampling Page 内で，素数が配置される範囲においてビット反転が発生するレイアウトを 1 つ作成する．

step1-2. 5 章の方法で測定した Dummy Page 数を用いて，Target Page 上に秘密情報を誘導する．

step1-3. レイアウトに対して Rowhammer を実行し，Sampling Page 内の step1 でビット反転が発生した位置のビット値を読み取る．

step2. step1 で得られた秘密情報に含まれる誤りを，Paterson らのアルゴリズムで訂正し，秘密情報の部分情報を復元する．

step3. 復元した秘密情報の部分情報から，Coppersmith のアルゴリズムを用いて全てのビット値を復元する．

次に，この攻撃で秘密情報を復元する実験について説明する．復元する情報は 2048 ビットの RSA 秘密鍵を構成する 1024 ビットの素数 p, q とする．

まず，RAMBleed で読み取る必要のある情報の量を求めるために，Coppersmith らのアルゴリズムによる復元を事前実験として行った．その結果，1024 ビットの素数の下位 570 ビットが得られれば，素数の全てのビット値を復元できることが分かった．したがって，step1 では，素数 p と q の下位 570 ビットを RAMBleed により読み取ればよい．このときに含まれる誤りは step2 の Paterson らのアルゴリズムにより訂正することが可能である．

実際に，素数 p を読み取るために step1 を実行した結果を表 4 に示す．なお，時間の都合上，570 ビットのうち 433 ビットを読み取った結果となっている．復元に必要な 570 ビットは足りていないが，読み取りを続けることで必要なビット数を確保できることが確認できた．表中の Correct は，RAMBleed で読み取った 433 ビットのうち，正しい値として読み取ったビット数の割合を示す．また，$P(0 \to 1)$ は p において値が 0 であるビットのうち 1 として読み取られた割合，$P(1 \to 0)$ は p において値が 1 であるビットのうち 0 として読み取られた割合である．同じビット位置の値が複数回読み取られた場合，過半数以上の読み取り結果をそのビット位置の値とした．

次に，表 4 の誤り率のときに，Paterson らのアルゴリズムでその誤りを訂正できるかを確かめる．Paterson らのアルゴリズムにおける α,β を，それぞれ表 4 に示した $P(0 \to 1)$，$P(1 \to 0)$ の値とする．実験では，α,β の割合で p,q にランダムにビット反転を生じさせた値を，t=13,L=10 とし

て復元する試行を 100 回行った．実験の結果，64%の確率で復元に成功することが分かった．

以上の実験から，RAMBleed で読み取った情報から p,q の全てのビット値を復元できることが分かった．

ただし，Paterson らのアルゴリズムによる復元の成功率が 100%ではないため，RAMBleed で読み取った秘密情報から正しい秘密情報を必ず復元できるとは限らない．そのため，復元した部分情報から Coppersmith の手法で復元した秘密情報が正しくない場合，再度 RAMBleed で秘密情報を読み取る必要がある．

7. 制限事項

本論文では，OpenSSL を用いて SSL/TLS 通信に対応した Apache Web サーバに対して RAMBleed を実行し，サーバが保有する RSA 秘密鍵を構成する 2 つの素数を RAMBleed によって読み取り，復元できることを示した．しかし，次の理由で，本論文で示した攻撃が任意の実環境で成立するわけではない．

- Rowhammer によってビット反転が起こるビットの割合は，メモリの規格や個体によって変化する．もしビット反転する割合が小さいと，準備フェーズに要する時間が長くなる．さらに，ビット反転する割合が小さすぎると，秘密情報の回復に必要なビット数を得られなくなることもある．したがって，一定数以上ビット反転するメモリを使用する必要がある．
- 攻撃対象の Web サーバによっては，サーバプロセス以外のプロセスが稼働していたり，攻撃者意外のクライアントと通信する子プロセスが存在している可能性がある．そのため，秘密情報の誘導時に攻撃プロセスから解放したページが，攻撃者と TLS ハンドシェイクを実行しているプロセスではないプロセスに割り当てられる可能性がある．したがって，必ずしも本論文で示した精度で誘導が成功するとは限らない．
- Apache や OpenSSL の設定によっては，Dummy Page 数が本論文で示した値とは異なる可能性がある．Dummy Page 数が未知である場合，RAMBleed の実行時に指定する Dummy Page 数を調節しながら攻撃を繰り返すなどの操作が必要となる．

8. まとめ

DRAM の高密度化により，メモリ上でビット反転を生じさせる Rowhammer と呼ばれる攻撃が発見された．また，Rowhammer を利用してメモリ上のデータを間接的に読み取る RAMBleed が提案されている．RAMBleed は管理者権限を必要とせず，コンピュータ内で保持される秘密情報を高い精度で読み取ることができるため，非常に脅威性の高い攻撃である．本論文では，OpenSSL を用いて SSL/TLS 通信に対応した Apache Web サーバ上で RAMBleed を実行し，サーバが保有する秘密情報を読み取る方法を提案した．Apache および OpenSSL の挙動を解析した結果，RSA 秘密鍵を構成する 2 つの素数を高い確率で任意のメモリ位置に誘導できることが分かった．また，この結果を用いて RAMBleed を含む一連の攻撃を実装し，実験を行った結果，素数を復元できることを明らかにした．提案手法は，未だ回復手法が提案されていない OpenSSL で利用される秘密情報を RAMBleed を利用して回復する革新的な方法である．

謝辞　本研究の一部は JSPS 科研費 20K11810 の助成を受けたものである．

参考文献

[1] Y. Kim, R. Daly, J. Kim, C. Fallin, J. Lee, D. Lee, C.Wilkerson, K. Lai, and O. Mutlu, “Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors,” ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014, pp.361–372, 2014.

[2] M. Seaborn and T. Dullien, “Exploiting the dram rowhammer bug to gain kernel privileges,” Black Hat, vol.15, p.71, 2015.

[3] S. Bhattacharya and D. Mukhopadhyay, “Curious case of rowhammer: Flipping secret exponent bits using timing analysis,” Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings, pp.602–624, 2016.

[4] Y. Jang, J. Lee, S. Lee, and T. Kim, “Sgx-bomb: Locking down the processor via rowhammer attack,” Proceedings of the 2nd Workshop on System Software for Trusted Execution, SysTEX@SOSP 2017, Shanghai, China, October 28, 2017, pp.5:1–5:6, 2017.

[5] A. Kwong, D. Genkin, D. Gruss and Y. Yarom, ”RAMBleed: Reading Bits in Memory Without Accessing Them,” in 2020 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, US, 2020 pp. 695-711.

[6] 長濱拓季, 瀧田愼, 廣友雅徳, 森井昌克, “効果的な single-sided RAMBleed の提案,” 信学技報, vol. 119, no. 437, ICSS2019-91, pp. 145-150, 2020 年 3 月

[7] OpenSSL Software Foundation, “OpenSSL Cryptography and SSL/TLS Toolkit,” https://www.openssl.org (2021 年 8 月 23 日参照).

[8] The Apache Software Foundation, “The Apache HTTP Server Project,” https://httpd.apache.org (2021 年 8 月 23 日参照).

[9] K. G. Paterson, A. Polychroniadou, and D. L. Sibborn, “A coding-theoretic approach to recovering noisy RSA keys,” ASIACRYPT 2012. LNCS, vol. 7658, pp. 386–403. Springer, Heidelberg (2012).

[10] N. Heninger, H. Shacham, “Reconstructing RSA private keys from random key bits,” CRYPTO 2009. LNCS, vol. 5677, pp. 1–17. Springer, Heidelberg (2009).

[11] W. Henecka, A. May, A. Meurer, “Correcting errors in RSA private keys,” CRYPTO 2010. LNCS, vol. 6223, pp. 351–369. Springer, Heidelberg (2010).

[12] D. Coppersmith, “Small solutions to polynomial equations, and low exponent RSA vulnerabilities,” J. Cryptology, 10(4):233–260, 1997.