Vol.2022-ARC-248 No.60
Vol.2022-SLDM-198 No.60
Vol.2022-EMB-59 No.60
2022/3/11

CPU/GPU統合のSoCにおける サイドチャネル攻撃の検討及び検証

土谷 続季[1,a)]　木村 啓二[1,b)]

概要：車載コンピュータによる深層学習による物体検知処理など，組み込み分野でも GPU 等のアクセラレータを搭載した SoC が利用されるようになってきた．それらの SoC は 1 チップに実装するという制約からホスト CPU とアクセラレータが主記憶を共有する，CPU-GPU 統合 SoC の構成が多い．一方で，コンピュータの利用範囲の広がりからサイドチャネル攻撃等の脅威が組み込み機器でも問題となっている．本稿では，CPU-GPU 統合 SoC におけるサイドチャネル攻撃の可能性を検討し，GPU のメモリ使用量の変化を観測することによるユーザがアクセスしている Web サイトを特定する Rendered Insecure と，GPU カーネル間で Covert Channel を構築する攻撃が可能なことを確認した．また，Covert Channel については最大 505bps で転送が可能なことを確認した．

1. はじめに

自動運転技術の普及などに伴い，車載コンピュータにおける深層学習による物体検知処理など，組み込み分野においても高いコンピューティング性能が求められる場面が増えてきた．それに伴い，GPU 等のアクセラレータを搭載した SoC が利用されるようになってきた．これらの SoC は，小型化，省電力化のため 1 チップに実装するという制約からホスト CPU とアクセラレータが主記憶を共有する，CPU-GPU 統合 SoC（iGPU）の構成が多い．

このように組み込み機器においても GPU を利用可能な高性能システムが利用されるようになった一方で，コンピュータの利用範囲の広がりから，これらの機器は人々の生活に密着する多くの用途で使用され，またこれまで以上にプライバシーに関わる秘密の情報を扱うことになる。またそれに伴い，多くのセキュリティ上の攻撃にさらされるリスクが拡大している．これらの攻撃の中で，システムの挙動を観測することにより内部の秘密情報を解析するサイドチャネル攻撃は代表的な攻撃の一つである．

サイドチャネル攻撃に関して，分離 GPU（dGPU）環境や，Intel 製 CPU におけるオンボードグラフィックスでの攻撃はこれまで研究が行われている．しかしながら，組み込み用 iGPU 環境におけるサイドチャネル攻撃に関しては十分な調査が行われていない．

本稿では，CPU-GPU 統合 SoC におけるサイドチャネル攻撃の可能性を検討し，さらに NVIDIA Jetson Xavier NX[1] で実際にいくつかの手法で攻撃が可能なことを確認したので，その結果について報告する．まず、攻撃手法を CPU と GPU 間の攻撃，GPU 内部での攻撃の二種類に分類して検討を行った．CPU と GPU 間での攻撃では，GPU のメモリ使用量の変化を観測することによるユーザがアクセスしている Web サイトを特定する Rendered Insecure を Jetson 上でいくつかの Web サイトにより検証した [2]．dGPU に対して提案された本手法が，Jetson Xavier NX においても行えることが確認できた．また，GPU と GPU 間の攻撃では，GPU を利用している異なるプロセス間で Prime+Probe [3], [4], [5], [6], [7] を用いた Covert Channel の構築を行い，最大 505bps でのデータの転送が可能なことを確認した．

以下，2 節で dGPU や Intel 内蔵 GPU(iGPU) でのサイドチャネル攻撃に関する既存研究を概観し，3 節で Jetson Xavier NX 概要を説明する．4 節で実際の攻撃の手法の検討，5 節で Jetson Xavier NX における攻撃手法の実験結果を報告し，6 節でまとめる．

2. 関連研究

本節では，これまでに dGPU や Jetson 以外の iGPU 環境に対して提案されたサイドチャネル攻撃について述べる．

[1] 早稲田大学基幹理工学研究科
School of Fundamental Science and Engeneering, Waseda Univ
[a)] tsuzu@kasahara.cs.waseda.ac.jp
[b)] keiji@waseda.jp

Vol.2022-ARC-248 No.60
Vol.2022-SLDM-198 No.60
Vol.2022-EMB-59 No.60
2022/3/11

2.1 Covert Channel

Covert Channel は 2 つのプロセスなどの間で許可されていない通信路を確立することである．この時通信路を確立する 2 つのプロセスを Spy(送信側) と Trojan(受信側) と呼ぶ．NVIDIA dGPU におけるサイドチャネルでは GPU 内部の L2 や L3 キャッシュの Prime+Probe によりデータを転送する [6]． Intel iGPU における攻撃では，CPU と GPU で Last Level Cache（LLC）を共有しているため，この LLC を利用して Covert Channel を構築している [5]．Prime+Probe は受信側がメモリ上に確保したデータをキャッシュにロードし，そのデータがキャッシュから追い出されたこと (Flush) を検知し，追い出された時間や頻度の変化を計測することで 1 ビットずつデータを受信する．送信側はキャッシュを何らかの方法で Flush する必要がある．送信側における受信側データのキャッシュの Flush は，予め提供された API を利用したり，同じキャッシュにロードされる異なるデータを読み出し続けることで実現する．NVIDIA dGPU や Intel iGPU における Covert Channel では後者を利用している．また，NVIDIA dGPU ではデータキャッシュを利用した Covert Channel [6] とインストラクションキャッシュを利用した Covert Channel [7] が存在する．キャッシュを利用した Covert Channel を構築するためには Spy と Trojan でキャッシュを共有している必要がある．また，通信路の確立にあたってはキャッシュの構成を特定する必要がある．

2.2 Rendered Insecure

Rendered Insecure は GPU におけるメモリ使用量やパフォーマンスカウンタの変化を観測し，予め学習した分類器を利用して，Victim がアクセスしている Web サイト，また入力した内容について予測するというものである [2]．Web サイトは表示される要素によってメモリ確保の時間やサイズが異なる．また，同じブラウザを利用していればアクセスするたびに一定のメモリアロケーションの挙動を示すことを利用し，メモリ使用量の変化を観測することで Victim がアクセスしている Web サイトを特定する．また，GPU のパフォーマンスカウンタの値を観測することでも Web サイトごとの特徴により同様の攻撃が可能である．

2.3 タイミング攻撃

暗号プログラムは通常，実行中の情報漏洩を防ぐために，入力にかかわらず定数時間で処理を行うよう実装されている．しかしながら，マイクロアーキテクチャレベルでは入力に応じて異なる動作をとるため，GPU で並列に動作するスレッドが共有メモリにある同一のバンクにアクセスした際に起こるバンクコンフリクトの発生時間の観測 [8] や電力消費量の変化を観測する [9] ことで AES の共通鍵等を盗み出す攻撃が可能である．どちらの手法も NVIDIA の dGPU において検証されている．

3. Jetson Xavier NX

本節では，本稿がサイドチャネル攻撃の検討対象とする Jetson Xavier NX について，そのアーキテクチャと GPU で利用可能なメモリの種別について概要を説明する．

3.1 Jetson Xavier NX のアーキテクチャ

NVIDIA Jetson Xavier NX は，ARM AArch64 ベースの Camel コアと Volta GPU を持つ SoC を搭載したシングルボードコンピュータである．ARM コアは 6 コア，GPU の CUDA コアは 384 基利用可能である [1]．また，メモリは CPU と GPU で共有で LPDDR4x 8GB を備えている [10]．GPU は 16 基の CUDA Core が 1 つの Streaming Multiprocessor (SM) と呼ばれる単位でまとめられる．本 SoC は SM を 24 基搭載し，SM ごとに 128KB の L1 キャッシュ，GPU 全体で 512KB の L2 キャッシュを備えている．GPU 部分の概略図を図 1 に示した [11]． 表 1 に Jetson Xavier NX の仕様を示す [10], [12]． なお，システムソフトウェアは本稿において利用したバージョンを示している．公開されていない情報については，CUDA SDK でサンプルとして提供されている deviceQuery コマンド等を利用して取得した [13].

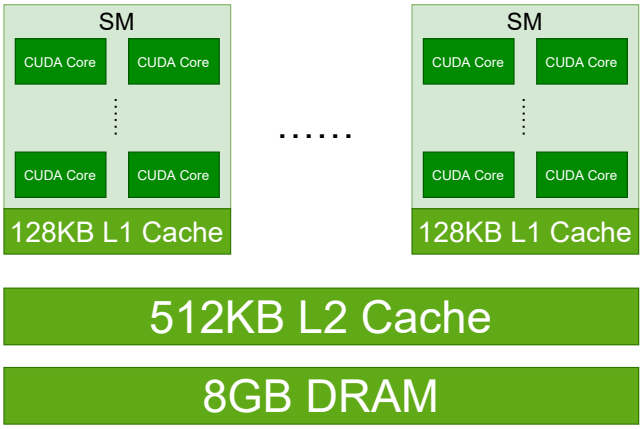

図 1 Jetson Xavier NX の GPU の構成図

3.2 Jetson Xavier NX の GPU メモリの種類

サイドチャネル攻撃の検討には対象となるシステムのメモリアーキテクチャの理解が必要である．以下で，Jetson Xavier NX の GPU で利用可能なメモリの種類である，Normal Memory，Unified Memory，及び Pinned Mmoery についてそれぞれ説明する（表 2 ）．

3.2.1 Normal Memory

GPU 専用に確保されており，CUDA において GPU で標準的に利用されているメモリのことを，本稿では Normal Memory と呼ぶ．Normal Memory は，dGPU 環境と

表 1 Jetson Xavier NX のスペック

CPU	アーキテクチャ	AArch64
	コア数	6
	L1d キャッシュ	64KB/Core
	L1i キャッシュ	128KB/Core
	L2 キャッシュ	2048KB/Cluster
	L3 キャッシュ	4096KB
	メモリ	8 GB(GPU と共有)
GPU	世代	Volta
	コア数	384
	L1 キャッシュ	128 KB/SM
		(一部 Shared Memory)
	L2 キャッシュ	512 KB
	メモリ	8GB(CPU と共有)
	CUDA Capability	7.2
システム ソフトウェア	JetPack	4.4
	OS	Linux4Tegra 32.4.3
		(Ubuntu 18.04.5)
	CUDA Runtime	10.2

表 2 Jetson Xavier NX の GPU で利用できるメモリ種別

	Normal	Unified	Pinned
アクセス可能	GPU	CPU/GPU	CPU/GPU
メモリ領域	独立	共有	共有
データコピー	必要	不要	不要
アドレス	独立	共有	共有
キャッシュ	GPU	CPU・GPU	GPU
コヒーレント制御	-	SW	HW/SW

同様に cudaMalloc 関数を用いて確保する．ただし，Jetson Xavier NX ではメモリは CPU と GPU で共有されているため，CPU と同じメモリ上に確保されるが，CPU から通常アクセスすることはできない．そのため，CPU 上のデータを GPU で処理するためには明示的に cudaMemcpy を利用して CPU がアクセス可能なメモリ領域から GPU のメモリ領域に対してデータを転送する必要がある．GPU 側では SM ごとの L1 キャッシュ 128KB と GPU 全体で共有の L2 キャッシュ 512KB を利用可能である．

3.2.2 Unified Memory

Unified Memory は CPU と GPU で共有して利用可能なメモリ領域であり，CPU と GPU でアドレス空間を共有する．そのため，CPU 側で確保したアドレスを直接 GPU カーネルに渡すことができる．本領域は cudaMallocManaged 関数を利用して確保される．また，CPU と GPU の両方で各々キャッシュを利用可能である．しかし，コヒーレント制御はソフトウェアで行われているため，ソフトウェアのバグなどで正常に制御が行われない可能性があり，重要なワークロードにおいては 3.2.3 小節 で説明する Pinned Memory を利用することが推奨されている [14]．また，Unified Memory はドライバによる高度な管理が行われているため，ページテーブルへのマッピング等でユーザからでは制御しづらい隠れた遅延が発生することがあり，パフォーマンスを最大に引き出せない場合もある [15]．

3.2.3 Pinned Memory

Pinned Memory は Page-Locked Memory とも呼ばれ，Jetson Xavier NX においては CPU と GPU どちらからもアクセスできるメモリ領域である．dGPU 環境においては CPU 向けに確保された領域を GPU から直接アクセス可能とするためのものであるが，Jetson Xavier NX では CPU と GPU が共有のメモリを利用しているため，通常のメモリとほとんど同じ形で利用できる．ただし，CPU からはキャッシュが利用できるものの，GPU からキャッシュを利用することができない．また，アドレス空間は独立で，cudaHostAlloc で確保して CPU でのアドレスを取得した後，cudaHostGetDevicePointer で GPU でのアドレスを取得する必要がある．

また，Jetson Xavier NX を含む CUDA Capability が 7.2 以上のデバイスでは I/O Coherency を持ち，Pinned Memory 利用時は GPU などの I/O デバイスは CPU キャッシュの最新の状態を取得できることがハードウェアによって保証されている．一方で，ハードウェアレベルで GPU が書き込んだ値を CPU で最新の値が取得するハードウェアレベルでの機構は存在せず，CUDA Driver によって実現されている．

この Pinned Memory のソフトウェアによるキャッシュのコヒーレンシー制御がいつ行われているかについては公開されていないため，簡単なプログラムによる検証を行った．検証に使用したプログラムは，GPU から Pinned Memory に対して 2 回の書き込みを行う間に一定時間のスリープを入れ，一方で CPU 側では Pinned Memory から直接値を読み出し，2 回の書き込みの間の経過時間を計測するものである．このプログラムを用いて，GPU でスリープを入れる時間とその時に CPU で観測される経過時間の関係について調べた．結果を表 3 に示す．表より，少なくとも GPU カーネルの実行中にもコヒーレンシー制御が行われており，GPU カーネルを実行しながらも Pinned Memory を経由して GPU から CPU にデータ転送が行えることがわかる．

表 3 スリープの時間と 2 回の Pinned Memory 書き込みの実経過時間の計測結果

スリープ時間 (クロック (GPU))	2 回の間の実経過時間 (クロック (CPU))
100,000	3,488
200,000	7,580
300,000	10,439

4. Jetson Xavier NX におけるサイドチャネル攻撃の検討

本節では，CPU/GPU 統合 SoC 搭載の Jetson Xavier NX

を対象とするサイドチャネル攻撃手法について検討する．本稿では CPU–GPU 間、及び GPU 内部のサイドチャネル攻撃を対象とし，特に CPU–GPU 間攻撃では Rendered Insecure を，GPU 内部攻撃では Covered Channel 構築を具体的に検討した．なお，CPU 内で完結するサイドチャネル攻撃は本稿の対象外とする．

4.1 CPU–GPU 間のサイドチャネル攻撃

4.1.1 Rendered Insecure

CPU–GPU 間のサイドチャネル攻撃として，まず 2.2 節で述べた Rendered Insecure について検討する．dGPU では CUDA API の cudaMemGetInfo 関数等を利用して GPU のメモリ空き容量を取得して攻撃を行なっていた．Jetson Xavier NX においても同様の API が利用可能であり，これを利用して dGPU と同様に Rendered Insecure が実施可能であると考えられる．この攻撃を実行するには攻撃対象のブラウザが実行中のデバイスで，CUDA のメモリ使用量取得 API を利用可能なプログラムを実行できる必要がある．

4.1.2 Covert Channel

Intel iGPU とは異なり，Jetson Xavier NX では CPU と GPU でキャッシュを共有していない．そのため，直接キャッシュを経由した Covert Channel を構築することはできない．しかしながら，3.2.2 節及び 3.2.3 節で述べたように，Unified Mmeory や Pinned Memory はハードウェアあるいは CUDA Driver によるキャッシュのコヒーレンシー制御が行われている．これを利用して CPU/GPU 間で Covert Channel の構築などが行える可能性がある．

4.2 GPU 内部でのサイドチャネル攻撃

Jetson Xavier NX は L2 キャッシュは全体で共有されるため，CUDA Kernel がどの SM で動くかに依らず L2 キャッシュを介したデータ転送が可能である．そのため，L2 キャッシュを利用した Covert Channel の構築を検討する．本方式は GPU 内部の処理となるため，Spy，Trojan 共に CUDA Kernel として CUDA Core 上で動作する．

Prime+Probe を利用したキャッシュ経由の Covert Channel の構築では，多くの場合キャッシュのライン単位で送信される．これにより，ノイズの削減や速度向上が期待できる．

しかし，Volta 世代の GPU では L1 キャッシュは Associativity が大きく，また Cache Replacement Policy が LRU ではないなど複雑な仕様が多い [16]． 加えてキャッシュ管理機構の詳細なアルゴリズムなどについては公開されておらず，特定のラインのみを利用することが難しい．そこで，本稿では L1 キャッシュの詳細な構成や挙動を特定することなく Prime+Probe を用いた Covert Channel の構築を試みた．また，GPU 上で一定時間のスリープを実行して転送する方式では Flush のキャッシュが意図した間隔で行われなかったため，CPU でスリープを実行し，都度 GPU カーネルを呼び出すようにした．

この攻撃は攻撃対象のデバイスにおいて，盗み出すデータを読めるが外部に転送を行うことができないプログラムと，外部に転送することはできるがデータを読むことができないプログラム等の間で通信チャネルを構築することを想定している．

4.2.1 送受信アルゴリズムの検討

前述の通り，L2 キャッシュの状態を利用して送受信することを検討する．

まず準備として，送信側・受信側の両方の CUDA Kernel で，L1 キャッシュサイズよりも大きく，L2 キャッシュサイズよりも小さい配列を GPU 上に各々確保する．

受信側では上記の配列に対し，一定のストライドで繰り返しアクセスを行うことで常に L2 キャッシュにアクセスするようにする．本配列アクセスの間，送信側の処理によって自配列の L2 キャッシュアクセスをミスした時，メモリアクセス時間の計測によりそれを観測可能である．そしてこのキャッシュミス発生の間隔の計測により，0 または 1 の受信を識別可能である．例えば，前回のキャッシュミスから今回のキャッシュミスまでの経過時間が 600μs 以下であれば 0，以上であれば 1 というようにデータが受信できる．

送信側は前述の配列全体に対して順番にアクセスを行うことで，その配列のデータを L2 キャッシュにロードすると共に，他プロセスの L2 上のデータを追い出す（Flush する）．この Flush の時間間隔を調整することで 0 または 1 として 1bit ずつ転送を行う．

受信側は一般に，キャッシュの Flush を検知した際にそれが送信側による意図的な Flush かを確認できない．そのため，送信側が送信を開始したことを通知するため，送信したいデータ（ペイロード）に先だって Ready to Send バイトを送信する．これは送信側と受信側で同じ値を利用できれば問題ないが，10101010 のような数ビットずらすと同じ値になるようなデータは避けるべきである．これは，Ready to Send バイトが送られる直前に他の要因の Flush で 10 が送られるたと受信側が判断すると，データ送信開始の境界が 2 ビットずれてしまうためである．本稿では 10110010 を利用した．受信側はこのビット列を受信するとその次のビットからがペイロードであると判断し，受信を開始する．送受信の流れを図 2 に示した．

4.2.2 Covert Channel 構築プログラムの実装

まず，送信側の実装について述べる．リスト 1 はキャッシュを Flush するために GPU でアクセスを行う配列を初期化する部分である．配列を 64 ビットの整数型にすると，次にアクセスするポインタを配列に保存しておくことができる．よって，リスト 2 に示す GPU 側のプログラムでは，

Vol.2022-ARC-248 No.60
Vol.2022-SLDM-198 No.60
Vol.2022-EMB-59 No.60
2022/3/11

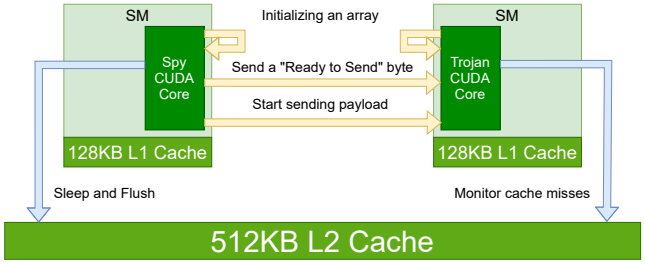

図 2 Covert Channel の実行の概略図

取り出した値をそのままポインタにキャストしアクセスを繰り返すだけでストライドアクセスによるキャッシュの Flush を行うことができる．最後に device_result に書き込まれた計算結果は利用していない値であるが，これを挿入することで計算結果への依存を生み，最適化等で処理が削除されないようにしている．

リスト 3 が送信するデータを CPU で制御している部分である．kernel は CUDA の関数で，キャッシュの Flush を行う．cudaDeviceSynchronize() は CUDA Kernel の実行を待機するために呼び出されており，CUDA Kernel が終了していないにも関わらず次の処理に移ってしまうことを防ぐ．また，前回の Flush からの経過時間を調整することでデータを送信するため，あらかじめ一度 kernel を呼び出す Flush を呼び出している．payload の最初の 1 バイトには Ready to Send バイトが保存されている．この data_to_send から 1 バイトずつ取り出し，内側ループで最下位ビットから順に転送している．ビットが立っているかどうかで usleep に渡す時間を調整することで Flush の間隔を調整している．

リスト 1 Covert Channel 送信側の CPU プログラムの配列初期化部分

```
int stride = 128 / sizeof(uint64_t);
for (int i = 0; i < array_size; ++i) {
    int t = i + stride;
    if (t >= array_size) t %= array_size;

    host_array[i] = (uint64_t)device_array + sizeof(
        uint64_t)*t;
}
```

次に，受信側の実装について述べる．受信側の GPU で動く CUDA プログラムをリスト 4 に示す．受信側は送信側のような都度の停止はせずに実行を続ける．受信したデータは CPU に転送するために Pinned Memory を利用している．Pinned Memory はキャッシュを介さずに GPU からアクセスするため，通信に利用している GPU のキャッシュを汚染せずに書き込み可能である．送信と同じように初期化された配列に対し，ストライドアクセスを行い，キャッシュが Flush されているかどうかを確認する．

リスト 2 Covert Channel 送信側の GPU プログラム

```
__global__
void kernel(uint64_t* device_array, volatile
     uint64_t* device_result) {
    volatile uint64_t* j = &device_array[0];

    for(int i = 0; i < 4096; ++i){
        j = *(uint64_t**)j;
    }

    *device_result = (uint64_t)j;
}
```

リスト 3 Covert Channel 送信側の CPU プログラムの送信部分

```
kernel<<<d, th>>>(device_array, device_result);
cudaDeviceSynchronize();

for (int idx = 0; idx < data_size; ++idx) {
    char value = payload[idx];

    for (int bit = 0; bit < 8; ++bit) {
        usleep(value & 1 ? WAIT_FOR_ONE :
            WAIT_FOR_ZERO);
        value >>= 1;

        kernel<<<d, th>>>(device_array, device_result
            );
        cudaDeviceSynchronize();
    }
}
```

per-multiprocessor counter により，GPU における現在のクロックの経過数を clock 関数を用いて取得でき，メモリアクセスの平均クロック数を計測できる．メモリアクセスが平均で 400 クロックを超えていた場合，L2 キャッシュが Flush されていると判定し，miss の値をインクリメントする．一方で下回っていれば hit の値をインクリメントし，結果を Pinned Memory 上の device_result に保存する．

GPU でミスやヒットが起こった時間を CPU で観測することで GPU カーネルで書き込まれた値を CPU で読みだすことができる．時間を測定するタイマーには CNTPCT_EL0 と呼ばれる AArch64 で利用可能なレジスタを利用して計測する [17].

5. 評価

本節では，4 節で検討した Rendered Insecure と Covert Channel を評価した結果について述べる．

5.1 評価環境

5.2 Rendered Insecure

5.2.1 評価手法

本稿で実施した Rendered Insecure では，Victim が Web サイトにアクセスしている間のメモリ使用量の変化を観測

Vol.2022-ARC-248 No.60
Vol.2022-SLDM-198 No.60
Vol.2022-EMB-59 No.60
2022/3/11

リスト 4　Covert Channel 受信側の GPU プログラム

```
__global__
void kernel(uint64_t* device_array, uint64_t*
     device_result, int array_size) {
    volatile register uint64_t * j = &device_array
        [0];
    register int miss = 0;
    register int hit = 0;

    for(;;) {
        volatile register int start = clock();
        for(int i = 0; i < 1280; ++i){
            j = *(uint64_t**)j;
        }
        volatile register int end = clock();

        uint64_t elapsed = end - start;
        if (elapsed > 400 * 1280) {
            ++miss;
        } else {
            ++hit;
        }

        device_result[0] = (uint64_t(miss) << 32) |
             hit;
        device_result[1] = (uint64_t)j;
    }
}
```

表 4　Rendered Insecure の評価環境

ハードウェア	Jetson Xavier NX
OS (Ubuntu 18.04.5)	Linux4Tegra 32.4.3
JetPack	4.4
CUDA Version	10.2
Web ブラウザ	Chromium 91.0.4472.101

し，Web サイトによってメモリ使用量の変化に特徴が現れるかどうかを確認した．検証した Web サイトは以下の 4 つである．

- https://amazon.co.jp
- https://youtube.com
- https://yahoo.co.jp
- https://google.co.jp

それぞれの Web サイトに対して 3 回アクセスを行い，アクセス時に cudaMemGetInfo 関数を繰り返し呼び出し続け，Web サイトをロードしている間のメモリ使用量の変化を観測した．

5.2.2　評価結果

図 3 に実験の結果を示した．横軸は何回目の cudaMemGetInfo 関数の呼び出しであるかを表し，時間の経過を表している．ロード開始時点を 0 とし，ロード終了時点までの変化を計測している．縦軸は計測開始時のメモリ使用量を 0 とした，メモリ使用量の変化を表している．

それぞれ 3 回実行しても変化しない，Web サイトごとに異なる特徴的なメモリ使用量の変化があり，Rendered Insecure 攻撃は可能であることが確認できた．これは Web サイトごとにロードしている画像やスクリプトが異なり，またそのロードする順序も異なるためにこのような変化が起きているためである．その一方で google.co.jp のような画像等の少ない軽量の Web サイトではロードもすぐに終わってしまうために判断に使える特徴も少ないために攻撃も難しいと考えられる．

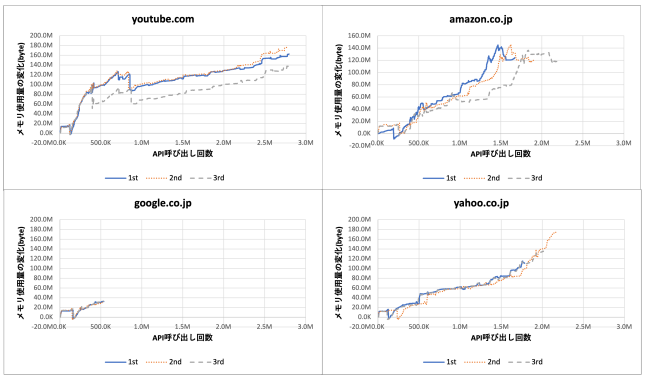

図 3　Rendered Insecure の評価結果

5.3　Covert Channel

5.3.1　評価手法

Spy と Trojan の GPU カーネルをそれぞれ別プロセスから起動し，0 と 1 が交互に連なる 128 バイトのデータを Spy から Trojan に対して送信した．Trojan 側で，Ready to Send バイトを受信してから，128 バイトを受信するまでにかかる時間を計測した．また，そのデータのうち誤っているビットの割合を求めた．本攻撃の Covert Channel では，Flush を実施する間隔を調整し，1 ビットずつ転送する．そのため，スリープの時間を変化させ，その時のエラー率と所要時間について調査した．以降 1 を送る際に待機した時間を W_1，0 を送る際に待機した時間を W_0 とし，単位は μs とする．本評価においては $W_1 > W_0$ となるように設定した．

5.3.2　評価結果

W_1 と W_0 を調整した際の Covert Channel のエラー率及び転送時間を 表 5 に示した．128 バイトのデータを送信によるエラー率 (E[bps]) と転送時間 (T[s]) から，転送速度 S[bps] を以下の式 (1) で求めることができる．その結果も表 5 に示した．

$$S = \frac{128 \times 8}{T} \times (1 - E) \tag{1}$$

また，横軸を W_0 とし，線の種類が W_1 表す時の実行時間，エラー率及び転送速度の結果を図 4，図 5，及び図 6 にそれぞれ示した．W_1 を 1200 未満にした場合の評価も行ったが，この場合はデータの転送が全て失敗となった．表及

び図中，値が存在しない部分はデータが転送できなかったことを示している．上記の結果より，Covert Channel を利用し，最大 505bps で転送できることが確認できた．

W_1 が低い場合や，W_1 と W_0 が近い場合に通信が不安定になり，エラー率が上昇したり通信が失敗するなどが発生していることがわかる．また，スリープの時間が伸びるほど転送速度も遅くなっていることがわかる．W_1 を 1500，W_0 を 1000 とした際に着目すると，0 と 1 を交互に送信した場合，平均 1250 μs のスリープが挿入される．この場合，理想的には 800bps で転送できることになるが，実際は 478bps しか出ていない．これは，この攻撃では GPU Kernel の呼び出しを 1 ビットを送るたびに行い，Flush を実行しているため，その呼び出し及び実行にかかる時間が存在するためである．

表 5 W_1 を変化させた際の Covert Channel の実験結果

W_1 ([μs])	W_0 ([μs])	エラー率	転送時間 ([s])	転送速度 (bps)
1200	600	0.38	2.02072	315.73
	700	0.14	1.98466	441.39
	800	0.16	1.96807	436.98
	900	0.20	1.97968	415.22
	1000	0.18	1.98738	423.17
	1100	-	-	-
	1200	-	-	-
1300	600	0.00	2.02863	504.77
	700	0.16	2.05503	418.97
	800	0.00	2.02388	505.46
	900	0.00	2.02958	503.55
	1000	0.00	2.02866	502.80
	1100	0.39	2.07075	303.27
	1200	0.38	2.12804	297.93
1400	600	0.00	2.0728	494.02
	700	0.00	2.08593	490.91
	800	0.00	2.08165	490.48
	900	0.00	2.08716	490.14
	1000	0.00	2.07595	492.79
	1100	0.01	2.11989	480.21
	1200	0.03	2.18479	452.67
1500	600	0.00	2.14062	478.37
	700	0.00	2.12263	482.42
	800	0.00	2.13112	480.50
	900	0.00	2.12585	481.69
	1000	0.00	2.14034	478.43
	1100	0.00	2.16942	472.02
	1200	0.15	2.21229	392.81

6. まとめ

本稿では，CPU と GPU が統合のアーキテクチャの組み込みシステムにおけるサイドチャネル攻撃の手法について検討を行い，Jetson Xavier NX 上で攻撃の検証を行った．

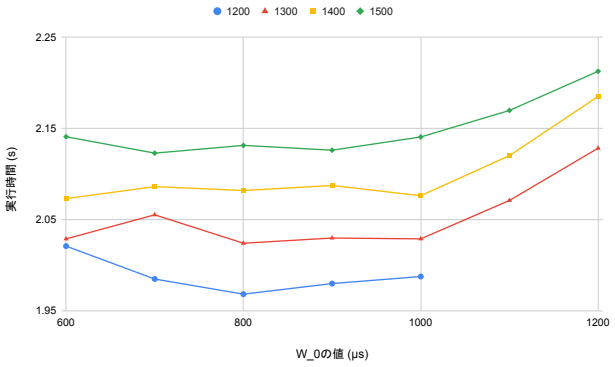

図 4 W_1 を変化させた際の Covert Channel の所要時間

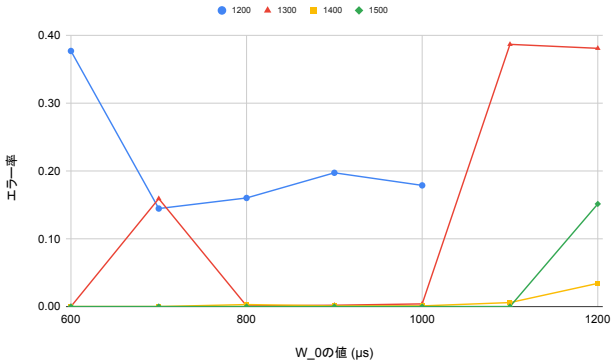

図 5 W_1 を変化させた際の Covert Channel のエラー率

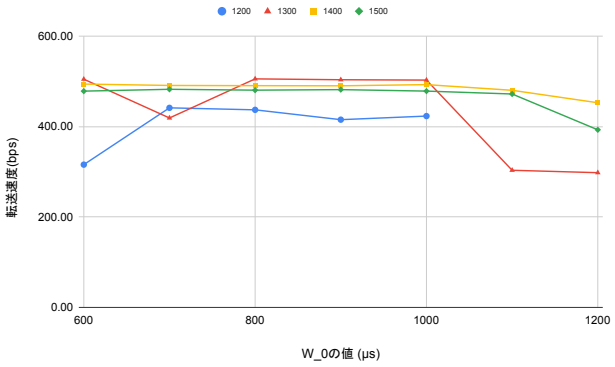

図 6 W_1 を変化させた際の Covert Channel の転送速度

その結果，CPU–GPU 間攻撃である Rendered Insecure，及び GPU 内部の Covered Channel 構築の二種類の攻撃について，実際に攻撃を行うことが可能であることを確認できた．

Rendered Insecure では，GPU におけるメモリ使用量の変化などを観測することで，Victim がアクセスしている Web サイトなど，Victim の行動を特定する攻撃であり，Jetson Xavier NX においても dGPU と同様の API を利用してユーザがアクセスしている Web サイトを特定することは可能であるとが確認できた．

GPU 内部における Covert Channel では，Jetson Xavier

NX の GPU の共有 L2 キャッシュを経由して 2 つの GPU カーネル間でデータを転送する Coveret Channel を構築した．初めに，受信側は一定サイズのデータに対して繰り返しアクセスを行い，常にそのデータが L2 キャッシュ上にある状態を保つ．また，その際受信側は平均アクセス時間を計測し，L2 キャッシュ上にデータがあるかを確認する．送信側は，同様に L2 キャッシュに対してより大きな容量を読み込むことで受信側のデータを L2 キャッシュから追い出す Flush 操作を行う．送信側は，Flush を行うインターバルを調整することでデータを送信する．インターバルの値を調整しつつ転送にかかる時間やエラー率を調整したところ，最大 505 bps で転送できることを確認した．

参考文献

[1] NVIDIA Corporation: 組み込み/エッジ システム用の Jetson Xavier NX, https://www.nvidia.com/ja-jp/autonomous-machines/embedded-systems/jetson-xavier-nx (Accessed: 2022-01-23).

[2] Naghibijouybari, H., Neupane, A., Qian, Z. and Abu-Ghazaleh, N.: Rendered Insecure: GPU Side Channel Attacks Are Practical, *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, New York, NY, USA, Association for Computing Machinery, p. 2139–2153 (online), DOI: 10.1145/3243734.3243831 (2018).

[3] Liu, F., Yarom, Y., Ge, Q., Heiser, G. and Lee, R. B.: Last-Level Cache Side-Channel Attacks are Practical, *2015 IEEE Symposium on Security and Privacy*, pp. 605–622 (online), DOI: 10.1109/SP.2015.43 (2015).

[4] Kayaalp, M., Ponomarev, D., Abu-Ghazaleh, N. and Jaleel, A.: A high-resolution side-channel attack on last-level cache, *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6 (online), DOI: 10.1145/2897937.2897962 (2016).

[5] Dutta, S. B., Naghibijouybari, H., Abu-Ghazaleh, N., Marquez, A. and Barker, K.: Leaky Buddies: Cross-Component Covert Channels on Integrated CPU-GPU Systems, *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pp. 972–984 (online), DOI: 10.1109/ISCA52012.2021.00080 (2021).

[6] Naghibijouybari, H. and Abu-Ghazaleh, N.: Covert Channels on GPGPUs, *IEEE Computer Architecture Letters*, Vol. 16, No. 1, pp. 22–25 (online), DOI: 10.1109/LCA.2016.2590549 (2017).

[7] Nayak, A., Ganapathy, V. and Basu, A.: (Mis) managed: A Novel TLB-based Covert Channel on GPUs, *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, pp. 872–885 (2021).

[8] Jiang, Z. H., Fei, Y. and Kaeli, D.: Exploiting Bank Conflict-Based Side-Channel Timing Leakage of GPUs, *ACM Trans. Archit. Code Optim.*, Vol. 16, No. 4 (online), DOI: 10.1145/3361870 (2019).

[9] Luo, C., Fei, Y., Luo, P., Mukherjee, S. and Kaeli, D.: Side-channel power analysis of a GPU AES implementation, *2015 33rd IEEE International Conference on Computer Design (ICCD)*, pp. 281–288 (online), DOI: 10.1109/ICCD.2015.7357115 (2015).

[10] NVIDIA Corporation: Jetson Xavier NX — NVIDIA Developer, https://developer.nvidia.com/embedded/jetson-xavier-nx (Accessed 2022/01/23).

[11] NVIDIA Corporation: NVIDIA Jetson AGX Xavier Delivers 32 TeraOps for New Era of AI in Robotics — NVIDIA Developer Blog, https://developer.nvidia.com/blog/nvidia-jetson-agx-xavier-32-teraops-ai-robotics/ (Accessed 2022/01/23).

[12] Schor, D.: Hot Chips 30: Nvidia Xavier SoC – WikiChip Fuse, https://fuse.wikichip.org/news/1618/hot-chips-30-nvidia-xavier-soc/ (Accessed 2022/01/23).

[13] NVIDIA Corporation: CUDA Samples :: CUDA Toolkit Documentation', https://docs.nvidia.com/cuda/cuda-samples/index.html#device-query (Accessed 2022/01/23).

[14] NVIDIA Corporation: https://docs.nvidia.com/cuda/cuda-for-tegra-appnote/index.html (Accessed 2022/01/27).

[15] Wang, Z., Wang, Z., Liu, C. and Hu, Y.: Understanding and Tackling the Hidden Memory Latency for Edge-based Heterogeneous Platform, *3rd USENIX Workshop on Hot Topics in Edge Computing (HotEdge 20)*, USENIX Association, (online), available from ⟨https://www.usenix.org/conference/hotedge20/presentation/wang⟩ (2020).

[16] Jia, Z., Maggioni, M., Staiger, B. and Scarpazza, D. P.: Dissecting the NVIDIA Volta GPU Architecture via Microbenchmarking, (online), available from ⟨http://arxiv.org/abs/1804.06826⟩ (2018).

[17] Arm Limited: Arm Armv8-A Architecture Registers, https://developer.arm.com/documentation/ddi0595/2020-12/AArch64-Registers/CNTPCT-EL0--Counter-timer-Physical-Count-register (Accessed 2022/01/23).