CHERI を用いた実行コンテキストに基づく In-memory KVS のメモリ隔離

西澤 彩花[1,a)]　山田 浩史[1,b)]

概要：In-memory KVS は，頻繁にアクセスされるデータをメモリにキャッシュすることで，アクセス速度を向上させることができる．In-memory KVS の代表例として Redis や Memcached があり，これらは大規模な Web サービスで使用されている．しかし，In-memory KVS はバグが発生しやすいという問題点がある．バグによって不正なアクセスが可能になると，攻撃者が任意の場所のデータの読み書きを行い，データの漏洩やサービスの中断を引き起こす．また，In-memory KVS はクラッシュした際にメモリ上のデータがすべて失われるという問題点がある．データサイズが大きいため，不揮発性メモリを使用したデータの復旧にも時間がかかる．本研究では，CHERI を用いてポインタアクセス時に実行コンテキストに基づいてアクセス権限を変更するメモリ隔離手法を提案する．CHERI とは，整数ポインタの代わりに偽造不可能かつ境界や権限情報を持つケイパビリティを使用することでメモリ保護を行うハードウェア支援機構である．ケイパビリティを使用してアクセス権限や境界を制限することで，不正なアクセスを防止する．提案手法は Redis 8.6.2 をベースにして実装を行った．実験の結果，5% 未満のオーバヘッドでメモリ隔離を行えることを示した．

キーワード：CHERI，In-memory KVS，memory isolation

1. はじめに

In-memory KVS（In-memory key-value store）は，頻繁にアクセスされるデータをメモリにキャッシュすることでアクセス速度を向上させるミドルウェアである．In-memory KVS は，Facebook [1] や Amazon [2] など，幅広い企業で実際に使用されている [3]．例として，Twitter（現 X）では，ストレージのキャッシュ，計算のためのキャッシュ，一時的なデータのためのキャッシュとして利用されている [4]．

しかし，In-memory KVS にはセキュリティと可用性の面で問題点が存在する．セキュリティの面では，ソフトウェアの脆弱性を利用して，保持しているデータの漏洩や改ざん，サービスの停止につながる恐れがあるという問題点がある．例として CVE-2025-49844 [5] がある．これは，In-memory KVS の 1 つである Redis [6] 上に存在していた脆弱性であり，スクリプトの機能を持つ Redis の全バージョンが影響を受けていた [7]．この脆弱性を悪用することで，攻撃者はサーバ上で任意の処理を実行することができる．具体的には，認証済みのユーザである攻撃者が悪意のあるスクリプトを作成し，サーバ上で実行させる．このスクリプトにより，メモリ管理を担う機構であるガベージコレクタが不正に操作される．その結果，本来アクセスされるべきでないメモリ領域が読み書きされ，最終的に任意の処理の実行が可能になるという仕組みである．In-memory KVS は，1 つのノードが複数のクライアントのデータを保持するため，こうした脆弱性による被害は 1 つのクライアントにとどまらず，同じノード上の他のクライアントも影響を受ける可能性がある．

可用性の面では，障害が起きた際に復旧に長い時間を要するという問題点がある．In-memory KVS はデータをメモリ上に保持するため，障害やバグによって再起動が必要になるとメモリ上のデータはがすべて失われる．この際，In-memory KVS のデータをディスク上にも保存しておき，再起動の際にこのデータを使用することで復旧できるが，データの読み込みとメモリ上の形式への変換が必要なため時間がかかる．例として，Facebook のサーバでは，1 台当たり約 120 GB のデータをディスクから復旧させるために最大 3 時間を要する [8]．

そこで，本論文では CHERI を用いてポインタアクセス時に実行コンテキストに基づいてアクセス可能範囲と権限

[1] 東京農工大学
Tokyo University of Agriculture and Technology
[a)] nishizawa@asg.cs.tuat.ac.jp
[b)] hiroshiy@cc.tuat.ac.jp

ⓒ2026 Information Processing Society of Japan

を設定するメモリ隔離手法を提案する．実行するコマンドに基づいてアクセスできるオブジェクトを変更することで，本来アクセスされないオブジェクトの読み書きを防止する．具体的には，実行するコマンドで読み書きが発生するオブジェクトにはロードとストア権限を，読み込みが発生するオブジェクトにはロード権限のみを，それ以外のオブジェクトはロードとストア権限のいずれも持たない状態で保持しておくことで，不正なアクセスが行われないようにする．

提案手法では，CHERI（Capability Hardware Enhanced RISC Instructions）と呼ばれるメモリ隔離のためのハードウェア機構を使用する．CHERI では，ポインタを偽造不可能なケイパビリティに置き換えてメモリアクセスを行うことが可能になる [9]．ケイパビリティにはアクセス可能な範囲が指定されており，範囲外にアクセスすることはできない．ハードウェアはケイパビリティを介する全てのメモリアクセスにおいて境界を検査し，範囲外のアクセスを例外として検出する．ケイパビリティにはパーミッションが含まれており，ロードやストア，命令フェッチなどの使用を制限できる．ケイパビリティの偽造不可能性は，ケイパビリティサイズの各メモリ位置にあるタグによって保障される．タグが有効なときのみケイパビリティが持つアドレス先にアクセスすることができる．通常の書き込みはタグが無効化されているため，データからケイパビリティを作り出すことはできない．これにより，不正にポインタを作成することができず，アドレスと値を明確に区別することが可能である．

提案手法では，CHERI を使用することでポインタのアクセス範囲と権限を制限できる点に注目する．CHERI を利用して，`GET` や `SET` のコマンドの処理を行う際に使用される整数ポインタをケイパビリティに置き換える．使用されるケイパビリティはロードとストアの権限が取り除かれ，アクセス範囲を制限された状態で保持される．コマンドの実行中，ロードやストアを行う時のみケイパビリティに権限を付与することで，範囲外のアクセスや不当なポインタによる値の読み書きを防ぐ．

プロセス内でメモリ隔離を行う手法である Intel MPK（Memory Protection Keys）[10] と比較した CHERI の利点として，メモリレイアウトを変更せずにより細粒度でメモリ保護を行えることが挙げられる．Intel MPK を利用したメモリ分離は，分離の単位が最小でもページごとであるため，アクセス権限を個別に制御したいオブジェクトは，同じ権限で扱うオブジェクトを同じ保護キーが割り当てられているページにまとめて配置する必要がある．一方，CHERI ポインタごとにアクセス範囲や権限を指定できるため，オブジェクトの再配置が不要である．

本論文の貢献は以下の通りである．

- CHERI を用いて実行コンテキストに基づく In-memory KVS のメモリ隔離手法を提案した．ケイパビリティはアクセス範囲と権限をあらかじめ制限しておき，必要な時のみケイパビリティのアクセス権限を変更することで，範囲外のアクセスや不当なポインタによって値が書き換えられることを防ぐ．
- 提案手法を実現するために必要なメカニズムを明らかにし，設計を行った．コマンド処理で使用されるポインタを特定し，メモリ確保時のメタデータを使用してケイパビリティのアクセス範囲を制限する．また，CHERI は hybrid-capability mode で実行し，特殊なケイパビリティである DDC を使用して権限を変更する．
- Redis 8.6.2 を用いて提案手法を実装した．`GET` や `SET` のコマンド処理で使用される整数ポインタをケイパビリティに置き換え，アクセス範囲や権限を制限した状態でケイパビリティを保持する．ケイパビリティによるアクセスが必要な時のみ，ケイパビリティに必要最小の権限を付与してアクセスを行う．
- 実験の結果，5% 未満のオーバヘッドでアクセス範囲の制限や権限変更を行えることを示した．

本論文では，第 2 章で In-memory KVS の具体例や問題点について例を用いて述べる．また，CHERI の技術的詳細について述べる．第 3 章では，メモリ隔離を用いた攻撃の緩和やプロセス内分離を行う関連研究について述べる．第 4 章では，本論文で提案する手法のアプローチや提案手法を実現するのに伴う課題について述べる．第 5 章では，提案手法を実現するためのケイパビリティのアクセス権限の変更やケイパビリティを保持するために必要な処理について述べる．第 6 章では，第 5 章を元に実装した機構について実際のコードを示しながら述べる．第 7 章では，実装した機構が有効であるかを評価する実験を行い，結果をもとに考察を述べる．第 8 章では，本研究のまとめと今後の課題について述べる．

2. 背景

2.1 In-memory KVS

In-memory KVS は広く用いられている一方，メモリ保護に関する問題が存在する．この問題点について，具体例を挙げて述べる．

2.1.1 メモリ安全性

In-memory KVS はメモリ安全性に関するバグを引き起こしやすいという問題点がある．メモリ安全性に関するバグとして，バッファオーバフローと use-after-free が挙げられる．バッファオーバフローとは，バッファにデータを書き込む際，バッファのサイズを超えて書き込むことで，隣接するオブジェクトのデータを書き換えてしまう現象である [11]．一方，use-after-free とは，プログラムが誤って解放済みのメモリ領域にアクセスしてしまうエラーである

ⓒ2026 Information Processing Society of Japan

[12]．あるメモリ領域にアクセスする際に使用していたポインタに対して，その領域が解放され，別の領域として再割り当てされた後にも使用することで意図しない挙動を引き起こす．

これらのバグは C/C++ などのメモリ安全でない言語で書かれたアプリケーションで発生しやすい．例として，MITRE が公開する 2025 年版の CWE Top 25 では，use-after-free（CWE-416）が 7 位，stack-based buffer overflow（CWE-121）が 14 位，heap-based buffer overflow（CWE-122）が 16 位に挙げられている [13]．また，Microsoft の製品において，毎年セキュリティ更新で修正される脆弱性のうち，約 70% がメモリ安全性に関連するものである [14]．Redis や Memcached は C で実装されているため，この影響を受ける．

バッファオーバフローや use-after-free を利用することで，攻撃者はデータの破壊やプログラムのクラッシュ，悪意のあるコードの実行が可能になる [11]．Redis でバッファオーバフローを引き起こす脆弱性の例として，CVE-2021-32761 [15] がある．この脆弱性は `BITFIELD` コマンドを利用して想定位置と異なる領域のデータを読み書きする．`BITFIELD` コマンドとは，Redis の string 型のデータをビットの配列として扱い，オフセットを指定してアクセスするコマンドである [16]．例として「`BITFIELD mykey GET u4 0`」の場合は，「`mykey`」をキーとするデータにおいて，ビットオフセットが 0 の場所にある 4 ビットの符号なし整数の値を取得することを表す．この脆弱性では，32 ビットシステム環境において，パラメータである `proto-max-bulk-len` を大きな値に変更した後，`BITFIELD` コマンドを実行することでオフセットの計算をオーバフローさせ，本来と異なる場所に値を書き込むことで攻撃が可能になる．

2.1.2 メモリ隔離

In-memory KVS はメモリ隔離が不十分であるという問題点がある．Redis は単一プロセスでコマンドの処理を実行しており，各ノードは，担当する複数のクライアントのデータを同じアドレス空間上に配置する．したがって，クライアントのデータ同士はメモリ隔離されていない．そのため，バグを利用することで他のクライアントのデータが取得される可能性がある．

メモリ上のデータが隔離されていないために被害が拡大する例として Heartbleed（CVE-2014-0160）[17] がある．この脆弱性では，攻撃者が細工したパケットを送信すると，サーバが本来読み出すべき範囲を超えてメモリを読み出し，その内容を応答として返してしまう [18]．その結果，攻撃者はプロセス上のメモリから，秘密鍵やパスワードなどの機密情報を取得できる．仕組みは以下の通りである．TLS には，接続時に相手が存在しているかどうかを確認する Heartbeat 機能が存在する．通常，Heartbeat のリクエストを受信すると，受け取ったデータをそのまま送り返して応答する．しかし，暗号ライブラリである OpenSSL における Heartbeat 機能の実装では，境界チェックが行われていなかった．そのため，攻撃者が Heartbeat のリクエストを送る際にペイロード長を実際のデータ量よりも大きく指定することで，通信相手のメモリ上に存在しているペイロードに隣接するデータを読み取ることが可能になる．

2.2 CHERI

CHERI（Capability Hardware Enhanced RISC Instructions）は，従来の命令セットアーキテクチャ（ISA）をハードウェア機構で拡張し，細粒度のメモリ保護と，スケーラブルなソフトウェアの区画化を可能にする [19]．ポインタを範囲と権限を持つケイパビリティに置き換えることで，C/C++ における範囲外アクセスなどの空間的なメモリ安全性の問題をハードウェアで検出できる．以降では，ケイパビリティの詳細と，CHERI が提供する 2 つのコンパイルモードである hybrid-capability mode と pure-capability mode について説明する．

2.2.1 Capability

ケイパビリティは，整数ポインタに代わる，ハードウェアによって保護される新しいデータ構造である．ケイパビリティによってメモリアクセス時に境界や権限のチェックが行われる．ケイパビリティの幅は基盤アーキテクチャの整数ポインタの 2 倍であり，64 ビットアーキテクチャなら 128 ビット，32 ビットアーキテクチャなら 64 ビットである．ケイパビリティのうち，半分はケイパビリティが指すアドレスを保持し，残りの半分には圧縮されたメタデータが入る．メタデータとして，境界や権限などが含まれる．さらに，ケイパビリティには 1 ビットの有効性を示すタグが付随する．タグとメタデータの詳細は次の通りである．

- タグ：対応するケイパビリティが有効であるかを示す．タグが無効なケイパビリティは，ケイパビリティが保持するアドレスへのアクセスを行うことができない．ケイパビリティを通常のデータで部分的または完全な上書きを行うと，メモリ上のタグが無効化され，使用できなくなる．タグはレジスタとメモリの両方で管理される．レジスタでは，レジスタ自体がタグを保持できるように拡張される．メモリ上では，通常のアドレス空間から見えない別の領域に保存され，ケイパビリティと同じ粒度で管理される．すなわち，タグとケイパビリティは一対一対応である．
- 境界：アクセスを許可するアドレス範囲の上限と下限であり，アドレスの冗長性を利用して浮動小数点に似た形式で圧縮されて格納される．
- 権限：ロードやストア，命令フェッチなどのケイパビリティの用途を制限する．

また，ケイパビリティの作成と使用には，以下のルール

が課される．

- 来歴の有効性（provenance validity）：有効なケイパビリティから明示的に命令を発行することによってのみ新しく有効なケイパビリティを作成可能であることを指す．
- ケイパビリティの単調性（capability monotonicity）：新しいケイパビリティを作成するときに派生元のケイパビリティの権限と境界を超えられないことを指す．すなわち，新しいケイパビリティを作成する際に，使用するケイパビリティが持つ権限以上の権限や，アクセス可能な範囲を越えたケイパビリティを作成することはできない．しかし，これは完全ではなく，例外処理などを行う際はこのルールを満たさない．
- 到達可能なケイパビリティの単調性（reachable capability monotonicity）：プログラムが実行中にどのような操作をしても，到達可能なケイパビリティの集合は，最初に与えられたものを超えて広がらないことを示す．しかし，ドメインの切り替えは例外である．

これらに加えて，ケイパビリティは意図性（intentionality）を持つ．これは，OS カーネルにケイパビリティを渡す場合もそのケイパビリティのみを使用することを指す．これにより，confused deputy 問題を防ぐ．confused deputy 問題とは，カーネルなどのより高い権限を持つプログラムが，ユーザ空間のプロセスなどの権限の少ないプログラムの代わりに処理を行う際に，過剰な権限を使用して本来許可されない処理を実行してしまう問題のことである．

CHERI では，レジスタの拡張も行われる．特徴的なレジスタとして，PCC（program counter capability）と DDC（default data capability）がある．PCC は PC（program counter）を拡張したものであり，命令フェッチの範囲を制限する．DDC は，整数アドレスによる通常のロードやストアに対してアクセスできる範囲と権限を定めるケイパビリティである．通常，DDC のアクセス範囲は仮想アドレス空間全体である．整数アドレスによるアクセスは，プログラムが指定しなくても暗黙的に DDC の範囲内に制限される．

2.2.2 Pure-capability mode

Pure-capability mode では，リターンアドレスやスタックポインタなど，暗黙のポインタを含むすべてのポインタでケイパビリティを使用する．この際，DDC は NULL に設定され，全てのポインタがケイパビリティを介してアクセスするようになる．整数ポインタによるアクセスを行うことはできない．

2.2.3 Hybrid-capability mode

Hybrid-capability mode は，既存のコードとの互換性の維持を目標としたものである．デフォルトでは，整数ポインタが使用され，DDC を用いてアクセス範囲を制限する．「__capability」アノテーションをつけたもののみ，メモリアクセス時にハードウェアによるチェックが行われる [20]．

3. 関連研究

これまでも，メモリ安全性に関するバグに対処するためにメモリ隔離を行う手法が複数提案されてきた．この章では，メモリ隔離を行う複数の例を紹介する．はじめに，Intel MPK などを用いてメモリ隔離を行う手法として FlexOS と ERIM を紹介する．次に，ARM のハードウェア機構を使用してメモリ隔離を行う手法として PeTAL を紹介する．最後に，CHERI に対してケイパビリティのタイプや操作を追加してメモリ隔離を行う手法として CAPSTONE を紹介する．

3.1 FlexOS

FlexOS では，OS の安全性と性能のトレードオフをビルド時に柔軟に選択できるようにすることを目的としている [21]．従来のモノリシックカーネルなどの OS では，設計の段階で使用するセキュリティ機構が厳格に決まっている．そのため，既存の OS に後から一部に安全なコードを継ぎ足すような修正を行っても，システム全体のセキュリティ向上が部分的であるという問題がある．例として，Linux のカーネルの安全性向上のために安全なプログラミング言語である Rust を導入しても，コードの大半は依然としてメモリ安全でない C 言語などで書かれたままになるため，メモリ安全性の向上は限定的である [22]．また，Linux のシステムコールである `pkey_mprotect()` を使用してハードウェアによるドメインの分離を OS レベルでサポートする手法である CubicleOS [23] では，ドメイン切り替え時にユーザモードとカーネルモードの切り替えが発生し，パフォーマンスが低下するという問題がある．FlexOS では，コンパイルやデプロイ時にアプリケーションの要件に合わせて様々なハードウェア分離機構やソフトウェア保護を柔軟に構成する手法を提案している．FlexOS では，既存アプリケーションとの互換性や性能も優れたライブラリ OS である Unikraft [24] をベースとし，OS のコンポーネントをどのコンパートメントに配置するか，コンパートメントごとにどの分離機構を使用するかをビルド時に選べるようにしている．FlexOS では，プロトタイプ実装において，ハードウェア分離機構として Intel MPK（Memory Protection Keys）[10] や EPT（Extended Page Tables）を使用してメモリ隔離を行った．Intel MPK でページテーブルエントリの未使用ビットにメモリ保護キーを割り当てることでアドレス空間内に最大 16 個の保護ドメインを作成できる．一方，EPT はハードウェア支援による仮想化機構である．EPT を使用した FlexOS では，コンパートメントごとに独立した VM（Virtual Machine）を生成してメモリ隔離を行う．Redis を対象にした実験において，Intel MPK を用いたコンパートメント化で最大 1.2M request/s

ⓒ2026 Information Processing Society of Japan

のスループットを達成した．

3.2 ERIM

この論文では，コンポーネント間の切り替えが非常に高頻度なアプリケーションでも，コンパイラを変更せずに標準の Linux カーネル上で，低オーバヘッドかつ強固なプロセス内分離を実現することを目的としている [25]．アプリケーションでは，1 つのバグがプロセス全体の機密性・完全性を壊すのを防ぐために，プロセス内でデータを分離する必要がある．例えば，Heartbleed [17] のような脆弱性から暗号鍵を守るために，鍵を扱うコンポーネントを他のコードから分離することが挙げられる．しかし，鍵へのアクセスが高頻度で発生する場合，分離したコンポーネント間の切り替えコストが無視できなくなるという問題がある．この問題に対処するための手法として，LwC（light-weight contexts）[26] は，コンポーネントごとに別のアドレス空間を割り当ててプロセス内分離を行うが，ドメイン切り替えにカーネルやハイパーバイザの介在が必要であり，オーバヘッドが大きい．他の手法として，SFI（Software Fault Isolation）[27] を使用した Native Client [28] がある．しかし，境界チェックによるメモリ分離手法のため，オーバヘッドが大きい．また，制御フローハイジャック防止のために CFI（Control-Flow Integrity）と呼ばれる制御フローの確認機構との併用が必要になる．ERIM は，x86 CPU の Intel MPK [10] とバイナリ検査を組み合わせた，プロセス内メモリ分離手法を提案している．ERIM では，Intel MPK を使用して，信頼コンポーネント T のメモリ（M_T）とそれ以外のメモリ（M_U）を別々の MPK ドメインに割り当てる。Intel MPK では，各ドメインに対するアクセス権を決定する PKRU レジスタを使用してアクセス検査をハードウェアで行い，権限の変更をユーザ空間で行うため，ドメイン切り替えを低コストで実行可能である．実験により，ERIM は 10 万回/秒の切り替えを 0.04% から 1.0% のオーバヘッドで実行できることを示した．

3.3 PeTAL

PeTAL は，メモリ破壊を利用した data-only attack を緩和することを目的としている [29]．Linux カーネルには多くのメモリ破壊バグが存在している．攻撃者はこれらを利用して data-only attack を行うことで，制御フローを変えずに UID や権限情報などの非制御データだけを書き換え，権限昇格や，本来アクセスできないカーネルリソースへの不正アクセスを実現できる．Data-only attack を防ぐ手法として，実行時の制御フローが静的に決まった制御フローグラフから外れないことを保証する CFI（Control-Flow Integrity）がある．Linux [30]，Android [31]，Microsoft Windows [32] などは CFI を導入しているが，データフローの正当性は保証されないため，これだけでは攻撃を防ぐことができない．Kenali [33] は，正規のデータフローだけを許す防御技術である DFI（Data-Flow Integrity）[34] を使用して data-only attack を防ぐ手法である．しかし，権限チェックで保護対象になるリソース自体を見つけられず，syscall 経由のパスである `modprobe_path` や ASLR の設定である `randomize_va_space` などの権限チェックを迂回して直接読み書きされるリソースを保護することができない [35]．PeTAL では，ARM の 2 つのハードウェア機能（MTE と PAC）を利用して data-only attack を緩和する手法を提案している．PeTAL では，はじめにシステムコールなどを使用して解析を行い，保護すべきカーネルのオブジェクトを特定する．保護すべきオブジェクトは MTE（Memory Tagging Extension）と PAC（Pointer Authentication Code）によって保護される．MTE [36] はメモリタグ機能であり，ポインタの上位バイトにタグを格納することで，メモリアクセス時にアクセスする領域のタグと一致するかを確認する．提案手法では，保護すべきリソースのみ非ゼロのタグをランダムに割り当て，それ以外にタグ 0 を割り当てることで，タグ 0 を持つ非特権ポインタからの不正アクセスを防止する．また，PAC [37] はポインタの認証機構である．ポインタ値などから暗号署名を計算してポインタ上位ビットに埋め込んでおき，ポインタの使用時に再計算して照合することで，ポインタの改ざんを防ぐ．実験において，PeTAL は NBench で 1.00 倍，LevelDB [38] で 1.03 倍の平均ランタイムオーバヘッドを示した．

3.4 CAPSTONE

CAPSTONE は，CHERI のケイパビリティの仕組みを拡張し，既存のハードウェアによる複数のメモリ保護モデルを単一のインタフェースで実現することを目的としている [39]．既存のハードウェアによるメモリ保護機能は，アーキテクチャごとに使えるものが異なっているという問題がある．例として，暗号鍵の生成や決済処理など，セキュリティが求められる処理を特別なメモリ領域で実行する TrustZone [40] は ARM プロセッサで使用でき，ポインタアクセス時に境界チェックを行ってバッファオーバフローを防ぐ Intel MPX（Memory Protection Extensions）[10][41] は Intel プロセッサで使用できる．このように各機能は特定の用途に特化しており，別の用途に柔軟に転用することが難しい．この問題に対し，様々なメモリ保護モデルを実現できる共通の基盤を作成するための候補として CHERI [42] がある．CHERI では，ポインタをケイパビリティに置き換え，ケイパビリティによってアクセス範囲を制限することで空間的メモリ安全性を提供する．しかし，CHERI には排他性がなく，一度他のドメインに渡したケイパビリティを確実に取り戻す手段がないため，時間的メモリ安全性を提供できない．また，メモリアロケータやス

ⓒ2026 Information Processing Society of Japan

ケジューラといった特権ソフトウェアを信頼する必要があり，トラストレスでない．CAPSTONE では，CHERI をもとに，排他的アクセスやケイパビリティの取り消しを同時に提供するトラストレスなメモリアクセスの基盤を提案している．linear capability と revocatoin capablity と呼ばれるケイパビリティの種類を追加し，revoke 操作を行えるようにすることで，排他的アクセスや他のドメインに渡したケイパビリティの回収操作を可能にする．実験では，CAPSTONE のエミュレータを実装し，アロケータやスケジューラを信頼しない形で実装できることに加え，3 種類のエンクレーブ型 TEE を実現できることを示した．また，gem5 [43] によるシミュレーションにより，SPEC CPU 2017 において，従来のシステムと比べたパフォーマンスオーバヘッドは 0% から 50% 程度に収まることを示した．

3.5 課題

本章では，関連研究として FlexOS，ERIM，PeTAL，CAPSTONE を紹介した．しかし，いずれの手法も課題が存在する．ここでは，紹介した各手法で残る課題について述べる．

Intel MPK や EPT などを使用して，メモリ隔離を行う手法として FlexOS と ERIM を紹介した．FlexOS はビルド時に使用するハードウェア分離機構やソフトウェア保護を選択して使用する手法である．ERIM はコンポーネント間の切り替えを行う処理において，強固な分離を低オーバヘッドで実現する手法である．これらはライブラリやコンポーネントの単位でプロセス内分離を行うことができる．しかし，これらの手法では，メモリ隔離の粒度の最小単位がページであり，オブジェクトごとに柔軟なメモリ隔離を行うことができないという問題点がある．また，Intel MPK ではメモリ隔離を行う際にメモリレイアウトを変更する必要がある．加えて Intel MPK で作成できるドメイン数は 16 個までであり，保護できる要素の数に限りがあるという問題がある．

ARM のハードウェア機構を使用してカーネル内のメモリ隔離を行う手法として PeTAL を紹介した．PeTAL では，MTE と PAC を使用することでカーネル内のオブジェクトを保護し，data-only attack を緩和する手法である．これにより，カーネル内の保護すべきオブジェクトに対する不正なアクセスを防止できる．しかし，MTE ポインタの上位バイトにランダムにタグの値が決定するため，オブジェクト同士でタグが衝突する可能性がある．したがって，この手法は確率的であり，完全なメモリ隔離を実現できていないという問題点がある．また，この手法は保護対象がカーネルオブジェクトであり，アプリケーションのオブジェクトは保護していない．

CHERI を利用してメモリ隔離を行う手法として，CAPSTONE を紹介した．CAPSTONE は CHERI に新しいケイパビリティタイプとケイパビリティの操作を追加することで，ARM の TrustZone や Intel の MPX など，複数のメモリ保護モデルを実現する手法である．これにより，エンクレーブ型 TEE によるメモリ隔離やトラストレスなアロケータの実装を行うことができる．しかし，これらのケイパビリティの機能をどのようにアプリケーションに適応させるかは述べられていない．また，権限を操作するアクセス制限は行っておらず，どのように活用するかは議論されていない．

4. 提案

本研究では，CHERI を使用し，実行コンテキストに基づいてポインタにアクセス可能範囲と権限を設定するメモリ隔離手法を提案する．

4.1 脅威モデル

本提案機構における脅威モデルは，In-memory KVS においてコマンドの実行中にメモリ安全性に関するバグを利用してオブジェクトのデータを読み取ったり，想定されていない書き込みを発生させることである．整数ポインタをケイパビリティ化することでコンテキストで想定されているアクセスのみを許可し，範囲外アクセスや想定されていないポインタの使用を禁止することで攻撃から保護することが本研究の目的である．

本研究では，コマンドの実行中に不正なアクセスが発生すると仮定する．一方で，カーネルの処理やコマンド以外の処理は正常に実行されると想定する．

本提案手法におけるデザインゴールを以下に示す．

- In-memory KVS の脆弱性最小化：実行コンテキストに応じてアクセスできるオブジェクトを制限することで，攻撃範囲を最小化する．また，アクセス範囲の制限により，悪意のある書き込みが他のオブジェクトに伝搬するのを防ぐ．
- メモリアロケータの変更が不必要：CHERI を利用したメモリ保護は，オブジェクトに対して個別に行う．ケイパビリティメタデータを使用してアクセス範囲や権限を制限するため，Intel MPK のようにページ単位で権限を設定する必要がなく，メモリレイアウトを変更しない．
- 未知の攻撃に対して耐性を持つ：提案手法では，コマンドの処理で使用されるオブジェクトのみにアクセス権限を付与するため，ホワイトリスト方式で制御する．そのため，将来新たな攻撃手法が発見されても，コマンドの処理に無関係なオブジェクトを使用する攻撃に対しては攻撃手法によらず防ぐことができる．
- 低いオーバヘッド：CHERI では，ハードウェアでアクセス範囲や権限のチェックを行うため，オーバヘッド

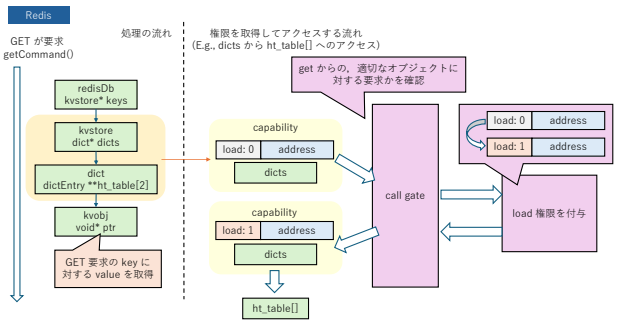

図 1 提案手法の全体像
Fig. 1 Overview

が小さい．また，権限の変更もケイパビリティに付随するメタデータの変更だけでよく，処理が軽量である．

4.2 アプローチ

In-memory KVS において，本来アクセスされないオブジェクトを利用したバグが多く存在する．2 章で述べた脆弱性などを利用して実行コンテキスト外のオブジェクトに対するポインタを取得した際，アクセスが制限されていなければそのデータを読み書きすることが可能になる．そこで，CHERI を使用してポインタのケイパビリティ化を行い，実行中にコンテキストに応じてケイパビリティが持つ権限を適切に操作する機構を提案する．実行コンテキストに応じてアクセス権限を切り替えることで，アクセス可能なオブジェクト数を削減し，不正なアクセスを防ぐ．これにより，たとえ脆弱性を利用して実行コンテキスト外のオブジェクトへのポインタを取得しても，権限不足によりアクセスが行えないようにする．ケイパビリティに置き換える整数ポインタは，コマンドの実行時にロードやストアの処理が発生するものに限定する．In-memory KVS である Redis の `GET` コマンド実行時に主に使用されるオブジェクトと処理の流れを例に提案機構を図 1 に示す．`GET` コマンドが実行された際，キーからハッシュ値を計算し，`RedisDB` の `keys`，`kvstore` の `dicts`，`dict` の `ht_table`，`kvobj` の `ptr` の順にオブジェクトをたどっていくことで対応するバリューを取得する．この時，`kvstore` 型のオブジェクトからケイパビリティ化されたポインタの `dicts` を利用して `dict` 型のオブジェクトにアクセスする状況に着目する．はじめは，`dicts` はロード権限とストア権限のいずれも持っていないため，このままでは `dict` オブジェクトのメンバにアクセスすることができない．そこで，`dicts` に対してロード権限を要求する処理を呼び出す．呼び出し先でケイパビリティにロード権限を付与する処理を行い，そのケイパビリティを使用して dict オブジェクトのメンバにアクセスを行う．ロードやストアが必要になった時のみ適切な権限を持つケイパビリティを使用し，それ以外ではロードとストアのいずれの権限も持たないケイパビリティを保持しておくことで，本来アクセスされないオブジェクトが読まれたり書き換えられたりすることを防ぐ．

ケイパビリティを利用して処理を行うのは `GET` や `SET` などのコマンドの処理のみである．コマンドの処理以外で行われるポインタアクセスは，全てケイパビリティのメタデータを取り除いた状態，すなわち整数ポインタを用いて実行される．全ての処理でケイパビリティを使用する場合，権限操作により実行時間が増加することが考えられる．なぜなら，保持されている全てのケイパビリティはロードとストアの権限が取り除かれているため，全てのケイパビリティアクセスにおいて権限を操作する処理を追加する必要があるからである．また，本研究ではコマンドの実行中の不正なアクセスを想定しているため，コマンドの処理以外の攻撃は考慮しない．したがって，コマンドの処理のみをケイパビリティに対応させることで十分である．

また，これに伴い，コマンドの処理でアクセスが発生する整数ポインタのみをケイパビリティに対応させる．コマンドの実行に必要なポインタのみをケイパビリティ化することで，コードの変更量を少なくする．しかし，使用する型のメンバごとにケイパビリティに置き換える有無を決定するため，コマンドの実行中にアクセスされないポインタがケイパビリティ化する場合がある．例として，`redisDb` のメンバである `expires` が挙げられる．`redisDb` には，他のメンバとして `expires` と同じ `kvstore*` 型である `keys` が存在する．`keys` はコマンド実行時に `keys->dicts` の順でアクセスされる．したがって，`kvstore` のメンバである `dicts` をケイパビリティに置き換える必要がある．この時，`expires` はコマンド実行時にアクセスされないが，`expires` が指す `kvstore` の `dicts` も同時にケイパビリティ化される．

4.3 Challenges

本提案機構を設計する上での課題を述べる．

- ケイパビリティに置き換える整数ポインタの特定：本提案機構を実現するにあたって，コマンドの実行で使用される整数ポインタをあらかじめケイパビリティに置き換える必要がある．また，使用されるポインタはコマンドごとによって異なる．したがって，コマンドごとに使用されるポインタをあらかじめ定義しておく必要がある．
- ケイパビリティの保持：本提案機構では，`GET` や `SET` などのコマンド実行中のみケイパビリティを使用し，それ以外ではケイパビリティを使用しない．しかし，コマンドの処理以外でもケイパビリティのロードやストアが発生する．例として，`GET` のコマンドでは `RedisDb` の `keys` にアクセスするが，サーバの起動時に `kvstoreCreate()` によって `kvstore` 型のオブジェクトを作成し，`keys` に書き込みを行う．その際，整

ⓒ2026 Information Processing Society of Japan

数ポインタをケイパビリティが想定されている領域に代入した場合はタグが 0 になり，今後ケイパビリティを利用したアクセスが行えなくなる．そのため，ケイパビリティ変数への書き込みを行う際は，整数ポインタをケイパビリティに変換し，アクセス範囲や権限を適切に制限してから書き込む必要がある．加えて，ケイパビリティをコマンド処理以外で使用する際は，メタデータを取り除き，整数ポインタとして扱う必要がある．

- アクセス権限の変更：ケイパビリティは，2.2.1 節で述べたケイパビリティの単調性の性質を持っているため，ロードやストアの権限を持たないケイパビリティからこれらの権限を持つケイパビリティを新しく作成することはできない．したがって，より高い権限を持つ他のケイパビリティを使用する必要がある．

5. 設計

本章では，提案機構を実現するための設計を述べる．In-memory KVS に提案機構を実装するため，Redis を使用し，GET コマンドの処理に対して設計・実装・評価を行う．また，CHERI を使用するにあたり，CHERI の機能を利用できる FreeBSD ベースの OS である CheriBSD を QEMU 上で動作させ，その上で Redis を動作させる．本章では，4.3 節で述べた課題に対する解決策について述べる．

5.1 ケイパビリティに置き換える整数ポインタの特定

Redis において，GET コマンドの処理で利用される整数ポインタをケイパビリティに置き換える処理を行う．本論文では，GET コマンドの処理として，getCommand() で呼び出される処理を指す．また，ケイパビリティに置き換える整数ポインタは，クライアント情報を保持する client 型のオブジェクトがアクセスできるポインタに対して行う．したがって，送信待ちのオブジェクトを保持する redisServer->clients_pending_write などは，GET コマンドの実行中にポインタを利用してアクセスされるが，client 型オブジェクトからアクセスされないため，整数ポインタである．

ケイパビリティに置き換える整数ポインタの特定は，uftrace を使用して行う．これにより，ユーザ空間の関数呼び出しをトレースしてコールグラフを取得可能である．その後，コールグラフをもとに，GET コマンドの処理でアクセスされるポインタを特定する．

5.2 コマンドの処理以外でのケイパビリティの保持

GET の処理以外でのケイパビリティへの書き込みは，整数ポインタを適切なケイパビリティに変換してから行う．具体的には，整数ポインタをアクセス範囲と権限が制限されたケイパビリティに変換する．整数ポインタをケイパビリティに変換する手順を図 2 に示す．はじめに，DDC を利用して，全ての範囲にアクセス可能かつロードとストアの権限を持つケイパビリティを作成する．次に，作成したケイパビリティからロードとストアの権限を取り除く．最後に，ケイパビリティのアクセス範囲を整数ポインタが指していたオブジェクトのサイズに制限する．

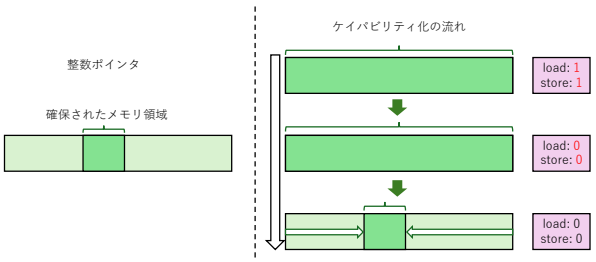

図 2 整数ポインタからケイパビリティへの変換フロー
Fig. 2 Integer-to-capability conversion process

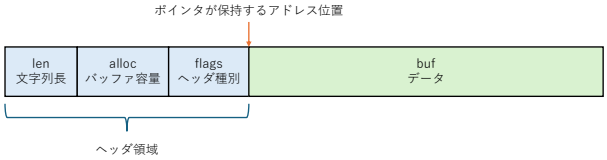

図 3 SDS の構造
Fig. 3 Structure of SDS

整数ポインタが指すオブジェクトのサイズの求め方は，オブジェクトのタイプによって異なる．以下に，タイプごとの処理について順に述べる．

malloc() で動的にメモリ確保され，ケイパビリティが持つアドレスがメモリ確保した領域の先頭を指している場合は，メモリ確保時のメタデータを使用してオブジェクトのサイズを求める．CheriBSD 上では，malloc_usable_size() を使用することでメモリ確保時のサイズを求める．malloc_usable_size() は，アロケータが各領域に付与して管理しているメタデータを参照し，その領域で実際に使用可能なサイズを返す関数である．引数にはアロケータが返した領域の先頭アドレスを渡す必要がある．malloc_usable_size() が使用できるポインタとして，kvstore 型のメンバである dicts が挙げられる．

ケイパビリティが持つアドレスがメモリ確保した領域の先頭ではないときは，アクセス範囲を制限せず，DDC のアクセス範囲と同じにする．例として，キーやバリューなどのデータを保持する SDS というデータ構造が挙げられる．SDS の構造を図 3 に示す．図のように，SDS の先頭には使用中の文字列長や文字列のバッファ容量などのヘッダ情報が配置されており，その後ろにキーやバリューのデータが存在する．また，SDS はヘッダ情報とデータ領域を同時にメモリ確保するが，SDS として扱われるアドレスは確保領域の先頭ではなくヘッダの直後，すなわちデータ領域

ⓒ2026 Information Processing Society of Japan

```
dictType dbDictType = {
    dictSdsHash,            /* hash function */
    NULL,                   /* key dup */
    NULL,                   /* val dup */
    dictSdsCompareKV,       /* lookup key compare */
    dictDestructorKV,       /* key destructor */
    NULL,                   /* val destructor */
    dictResizeAllowed,      /* allow to resize */
    .no_value = 1,          /* keys and values are unified (
     kvobj) */
    .keys_are_odd = 0,      /* simple kvobj (robj) struct */
    .keyFromStoredKey = kvGetKey,    /* get key from stored-
     key */
};
```

図 4 dictType 型オブジェクトの定義例
Fig. 4 Example of a statically defined dictType object

```
server.db[j].keys = kvstoreCreate(&kvstoreExType, &dbDictType
     , slot_count_bits, flags);
```

図 5 dictType 型変数の使用例
Fig. 5 Usage example of a dictType variable

の先頭を指す．したがって，SDS のアドレスをそのまま引数にして malloc_usable_size() を使用することはできないため，アクセス範囲を DDC と同じにする．

また，静的領域のオブジェクトに対するポインタでは，オブジェクトの型から推定したサイズでアクセス範囲を制限する．例として，dictType というデータ構造が挙げられる．dictType とは，ハッシュテーブルを持つデータ（dict, kvstore など）がキーやバリューをどのように扱うかを決める関数ポインタの集合である．dictType 型の変数の定義例を図 5 に，この変数を利用した処理の呼び出し例を図 4 に示す． 図 4 のように，変数 dbDictType はグローバル領域に静的に定義されている，また，図 5 のように kvstore の作成時にそのアドレスが渡されて参照されるだけであり，ヒープ上にメモリ確保は行わないため，malloc_usable_size() を使用できない．したがって，dictType のサイズは sizeof(dictType) で求め，求めたサイズでアクセス範囲を制限する．

5.3 アクセス権限の変更

ケイパビリティの権限を増やす処理は，DDC を使用して行う．例として，ロードとストアのいずれの権限も持たないケイパビリティに対してロード権限を増やす処理の流れを図 6 に示す．図の左側が元のケイパビリティのアクセス可能範囲と権限を，右側が新しく作成するケイパビリティのアクセス可能範囲と権限の推移を表している．以降は右側の流れについて説明する．はじめに，DDC を利用して新しいケイパビリティを作成する．DDC を使用して

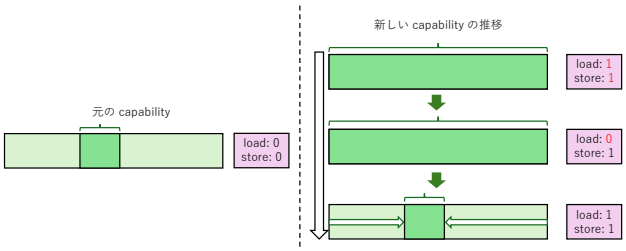

図 6 ケイパビリティの権限変更フロー
Fig. 6 Flow of capability permission modification

いるため，この時のアクセス範囲はプロセスのアドレス空間全体である．この時，ケイパビリティはロードとストアの両方の権限を持っている．次に，権限を制限する．今はロードのみを行うことを想定しているため，ストアの権限を取り除く．最後に，新しく作成したケイパビリティを元のケイパビリティのアクセス可能範囲に制限する．制限するサイズは，元のケイパビリティを利用することで容易に求められる．これにより，同じアクセス範囲でロード権限が付与されたケイパビリティを作成可能である．

一方，ケイパビリティの権限を減らす処理は，そのケイパビリティのみで行うことができる．

6. 実装

本章では，5 章で述べた設計をもとに Redis の実装について述べる．Redis [44] はバージョン 8.6.2 を使用した．はじめに，コマンド処理で使用する整数ポインタをケイパビリティに置き換える処理の実装について述べた後，コマンドの処理以外でケイパビリティを読み書きする処理の実装について述べる．最後に，アクセス権限を変更する処理の実装について述べる．

6.1 整数ポインタをケイパビリティに置き換える処理

Redis の GET コマンドでアクセス時に使用するポインタのうち，client 型のオブジェクトからアクセス可能な整数ポインタをケイパビリティに置き換えた．ケイパビリティに置き換える主な整数ポインタを GET コマンドの流れとともに表したものを図 7 に示す．図の右側が GET コマンドの処理の流れを示し，左側がそれに伴ってアクセスされるポインタを表している．また，付与されている番号は，実行する関数とアクセスするオブジェクトの対応関係を表す．GET コマンドでは，主にキーに対応するバリューを取得する際に，ポインタによるアクセスが発生する．そのため，キーやバリューのデータを保持する redisObject を指すポインタや，このデータを保持するハッシュテーブル ht_table を指すポインタ，ハッシュテーブルを複数のスロットに分けて保持する dicts を指すポインタなどをケイパビリティに置き換える必要がある．ケイパビリティ

ⓒ2026 Information Processing Society of Japan

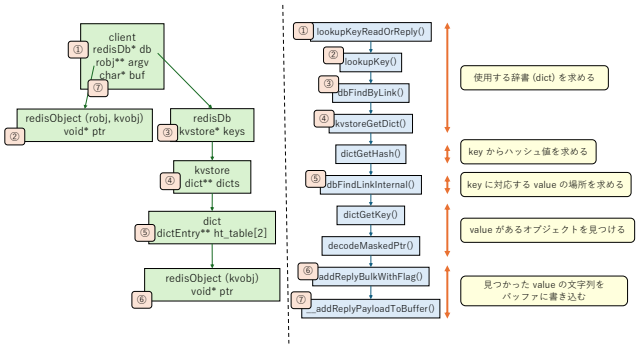

図 7 ケイパビリティに置き換える主要な整数ポインタ
Fig. 7 Primary integer pointers targeted for capability replacement

```
struct dict {
    dictType * __capability type;
    dictEntry * __capability *  ht_table[2];
        /* ... */
```

図 8 構造体メンバ内のケイパビリティ定義例
Fig. 8 Example of defining a capability as a struct member

に置き換える処理では，あるアクセス権限を持つケイパビリティが指すオブジェクトの間違ったメンバにアクセスしないようにするために，ケイパビリティに置き換える整数ポインタが存在する型は，全ての整数ポインタをケイパビリティに置き換える処理を行う．

また，整数ポインタをケイパビリティに置き換えたときのオブジェクトの定義例を，図 8 に示す．ポインタを示す「*」の後に「__capability」アノテーションをつけることで確保されるメモリサイズが整数ポインタの 2 倍になり，ケイパビリティのメタデータを入れることが可能になる．また，dict のメンバである ht_table はダブルポインタである．「dictEntry * __capability *」と記述することで，dictEntry* __capability 型のケイパビリティを持つ整数ポインタを定義することができる．「dictEntry * __capability * __capability」と記述すると dictEntry* __capability 型のケイパビリティを持つケイパビリティを定義することができる．しかし，これは ht_table にアクセスする際に 2 回の権限変更が必要になるため，ケイパビリティ化は dictEntry のオブジェクトを指すポインタのみに行った．

6.2 コマンドの処理以外におけるケイパビリティの読み書き処理

コマンドの処理以外でケイパビリティのデータを読みだす際は，コンパイラ拡張である「__cheri_fromcap」を使用する [45]．これを使用することで，明示的なキャストを行い，ケイパビリティを整数ポインタに変換することができる．ケイパビリティを整数ポインタに変換する

```
dict *kvstoreGetDict(kvstore *kvs, int didx) {
        return (__cheri_fromcap dict*)kvs->dicts[didx];
        }
```

図 9 ケイパビリティを明示的に整数ポインタに変換するコード例
Fig. 9 Example of explicit capability-to-integer pointer conversion

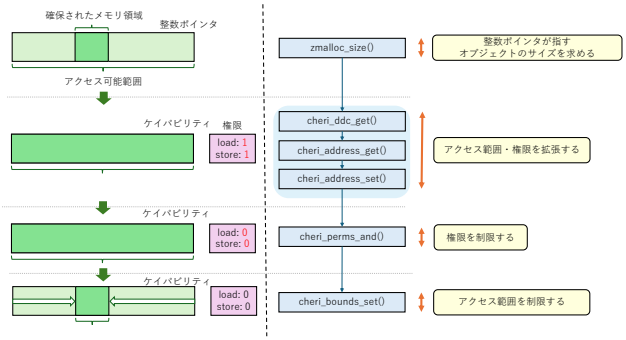

図 10 整数ポインタからのケイパビリティへの変換プロセス
Fig. 10 Process of converting an integer pointer to a capability

例を図 9 に示す．ここでは，kvstore 型のメンバである dicts のポインタを取得する．kvs は整数ポインタであるため kvs->dicts[didx] のアクセスができ，dict* __capability 型のケイパビリティを取得する．この取得したケイパビリティに対して「__cheri_fromcap dict*」でキャストすることで，dict* 型の整数ポインタになり，dict のメンバにアクセスすることが可能になる．また，kvstoreGetDict() の返り値を整数ポインタにすることで，ケイパビリティがプログラム内に広がるのを防ぎ，コードの変更量を抑えることができる．

一方で，コマンドの処理以外でケイパビリティに書き込む必要がある際は，整数ポインタをケイパビリティに変換し，アクセス範囲と権限を制限する．ケイパビリティに変換する際のアクセス範囲は，ポインタが持つアドレス位置によって異なる手法で求める．また，ケイパビリティに変換する際には CHERI 固有の関数を使用する．そのため，cheriintrin.h をインクルードして処理を記述した．

malloc_usable_size() が使えるオブジェクトにおいて，整数ポインタをケイパビリティに変換するプロセスを図 10 を用いて述べる．まず，zmalloc_size() によって malloc_usable_size() を呼び出し，ポインタが指すオブジェクトのサイズを求める．次に，DDC を使用して全アドレス空間にアクセスできるケイパビリティを作成する．DDC の呼び出しは cheri_ddc_get()，ケイパビリティが指すアドレスの変更は cheri_address_set() を使用することで行うことができる．次に，ケイパビリティの権限を制限する．権限は cheri_perms_and() を使用することで

ⓒ2026 Information Processing Society of Japan

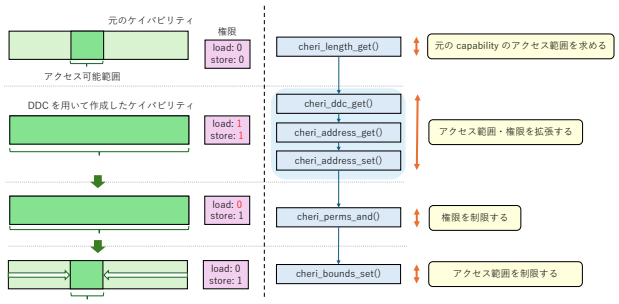

図 11 ケイパビリティの権限を増やすプロセス
Fig. 11 Process of elevating capability permissions

制限が可能になる．最後に，ケイパビリティのアクセス範囲を制限する．アクセス範囲は `cheri_bounds_set()` と，以前求めたオブジェクトサイズを使用することで制限することができる．以上の処理によって作成したケイパビリティを保持するための変数に書き込む．

ケイパビリティが持つアドレスがメモリ確保した領域の先頭でないときは，DDC を使用して新しいケイパビリティを作成したのち，`cheri_perms_and()` で権限のみを制限したものを書き込む．

また，静的領域のオブジェクトに対するポインタでは，はじめに，DDC を使用して新しいケイパビリティを作成する．その後，`cheri_perms_and()` で権限を制限し，`cheri_bounds_set()` にオブジェクトの型から推定したサイズを使用してアクセス範囲を制限したものを書き込む．

6.3 アクセス権限を変更する処理

コマンドの実行中，権限を持つケイパビリティが必要になった際に，ケイパビリティの権限を増やす処理を行う．ケイパビリティの権限を増やすプロセスを図 11 に示す．はじめに，`cheri_length_get()` を使用して元のケイパビリティのサイズを求める．この処理では 6.2 節と異なり，ケイパビリティには必ずアクセス範囲が指定されているため，ケイパビリティが持つアドレスの位置によって処理を変更する必要がない．これ以降の DDC を利用したケイパビリティの作成やアクセス範囲と権限の制限方法は 6.2 節と同様である．`cheri_ddc_get()` で DDC と同じ権限を持つケイパビリティを作成した後，`cheri_perms_and()` で権限を制限し，`cheri_bounds_set()` でアクセス範囲を制限する．

7. 実験

本章では，実装した提案手法に対して実験を行い，性能や効果を評価する．

7.1 実験環境

実験環境を表 1 に示す．

Redis は hybrid-capability mode で動作させるため，hybrid ABI でビルドした．CheriBSD は，cheribuild [46] を使用し，ターゲットを `run-morello-purecap` にしてビルドした．実験は CHERI 向けの QEMU を使用し，Arm morello をシミュレートして行った．

表 1 実験環境
Table 1 Experimental setup

構成	内容
CPU	Intel(R) Xeon(R) Gold 5418Y @3.8 GHz
メモリ	256 GB
ホスト OS	Ubuntu 24.04 (Linux kernel 6.8.0-136-generic)
Redis	バージョン 8.6.2
CheriBSD	2026 年 5 月 4 日時点での main ブランチ
QEMU	バージョン 7.0.0

表 2 オーバヘッド
Table 2 Overhead

Redis + CHERI（提案手法）	Redis（ベースライン）
5.89 ms	5.67ms

7.2 オーバヘッド

7.2.1 目的

本提案機構がどのくらい性能低下するのかを確認する．提案手法では整数ポインタをケイパビリティに変換する処理や，ケイパビリティの権限を増やす処理を追加しているため，オーバヘッドがかかることが予想される．したがって，元の性能からどれだけ時間が増加するかを調べる．

7.2.2 方法

クライアントがコマンドの実行を依頼してから応答が返ってくるまでにかかる時間を計測する．サーバにはキーが `sample`，バリューが `hello` のデータを一つだけあらかじめ登録しておく．クライアントとして，Redis に付属している redis-cli を使用し，「`GET sample`」のコマンドを送信する．このコマンドを 10000 回実行するのにかかる時間を計測し，得られた結果を実行回数で割ることで 1 回あたりにかかる時間を求める．比較対象はケイパビリティを使用しない Redis である．

7.2.3 結果

得られた結果を表 2 に示す．表より，提案手法の方がわずかに実行時間が増加していることが分かる．提案手法はベースラインよりも実行時間が 0.22 ms 増加し，割合にして 3.9% 実行時間が増加した．この増加時間は，主にコマンドの実行中に権限を変更する処理と，コマンドの実行以外で整数ポインタをケイパビリティに変換する処理が多くを占めていると考えられる．

7.3 不正なアクセスへの対処

7.3.1 目的

本提案機構が不正なアクセスに対して有効であるかを確

```
In-address space security exception (core dumped)
```

図 12 アクセス権限を持たないケイパビリティによるアクセス
Fig. 12 Memory access using a capability without sufficient permissions

```
Thread 1 received signal SIGPROT, CHERI protection violationwarning: could not convert 'si_code' from t
he host encoding (ISO-8859-1) to UTF-32.
This normally should not happen, please file a bug report.
.
Capability permission fault.
```

図 13 アクセス権限がないケイパビリティによるアクセス（GDB）
Fig. 13 Memory access using a capability without sufficient permissions (GDB)

認する．

7.3.2 方法

GET の処理を行う関数 getCommand() 内において，意図的に不正なアクセスを行うコードを実行させる．具体的には，本来アクセスを行わない領域へのアクセスを試みたり，範囲外アクセスを行ったりする．例えば，kvstoreDictFindLink() 関数内の kvstore* __capability 型のケイパビリティ kvs は本来ケイパビリティを利用したアクセスを行わないが，ここで「kvs->dicts」を行うことでアクセスを発生させ，権限の制限が適切であるかを確認する．また，dictFindLinkInternal() 関数では，dict* __capability 型であるケイパビリティの d にロード権限を付与してメモリアクセスを行うが，これに対して d[0x100] の値を一時変数にロードする処理を行うことで，オブジェクトのアクセス範囲が適切に制限されているかを確認することができる．このようなアクセスを行うことで，アクセス権限やアクセス範囲を適切に制限できているかを確認する．確認する対象のケイパビリティは，キーからバリューを取得する際に順にアクセスされる，client のメンバ db（redisDb* __capability 型），redisDb のメンバ keys（kvstore* __capability 型），kvstore のメンバ dicts（dict* __capability * 型），dict のメンバ ht_table（dictEntry* __capability * 型）とする．また，redisDb のオブジェクトは複数個がまとめてメモリ確保されるため，アドレスがメモリ確保した領域の先頭を指さないことがある．したがって，redisDb のオブジェクトを指すケイパビリティ（db）は権限の確認だけ行う．

7.3.3 結果

アクセス権限を持たないケイパビリティによるアクセスは全てにおいてエラーを検知し，図 12 のようにプログラムが停止した．また，GDB を使用して実行した場合は図 13 のような結果を得られた．GDB の結果から，権限の不足によるケイパビリティフォールトが発生していることが分かる．また，適切な権限を持つケイパビリティによる範囲外アクセスは全てにおいてエラーを検知し，権限不足の時と同様に図 12 の結果が得られた．また，GDB を使用して実行した場合は図 14 のような結果が得られた．GDB の結果から，許可されたアクセス範囲を越えたことによるケイパビリティフォールトが発生していることが分かる．以上により，ケイパビリティのアクセス範囲や権限を適切に制限できていると考えられる．

```
Thread 1 received signal SIGPROT, CHERI protection violationwarning: could not convert 'si_code' from the
host encoding (ISO-8859-1) to UTF-32.
This normally should not happen, please file a bug report.
.
Capability bounds fault.
```

図 14 適切な権限を持つケイパビリティによる範囲外アクセス（GDB）
Fig. 14 Out-of-bounds memory access by a capability with valid permissions (GDB)

8. おわりに

8.1 まとめ

本研究では，CHERI を用いて実行コンテキストに基づく In-memory KVS のメモリ隔離手法の提案を行った．CHERI がポインタのアクセス範囲や権限を制限できる点に着目し，これを In-memory KVS のメモリ隔離に使用する．提案手法では，コマンドの処理で使用されるポインタをケイパビリティに対応させ，必要な時のみケイパビリティの権限を増やすことで，範囲外アクセスや，不正なポインタによるアクセスを防ぐことが可能になる．提案手法を Redis の GET コマンドに対して行い，実験を行った．実験の結果，5% 未満のオーバヘッドでアクセス範囲の制限や権限変更を行えるという結果が得られた．また，不正なアクセスが行われた場合はそれを検知できたため，アクセス範囲や権限を適切に制限できていることを確認できた．

8.2 今後の課題

8.2.1 他のコマンドへの実装

Redis はメモリ削減のためにビットマスクを使用したり，メモリ確保したオブジェクトとは異なる型にキャストして処理を行ったりする場合がある．これらの処理が CHERI のケイパビリティのタグを意図せず無効化したり，範囲外アクセスを引き起こしたりしないかを GET コマンド以外もケイパビリティに対応させることで確認する必要がある．

8.2.2 細粒度のアクセス範囲の設定

本提案手法の実装では，メモリ確保した際の先頭アドレスを保持しない SDS や，様々な型のケイパビリティを保持する void* __capability 型において，アクセス範囲を DCC と同じサイズで実装を行った．より正確なメモリ確保を行うために，SDS に関する既存の関数の使用やコード解析を詳細に行うことで，これらのケイパビリティがアクセスできる範囲を制限することができる．

8.2.3 コールゲートの設置

ケイパビリティの権限を昇格する処理は，正当な要求のみ受理される必要がある．そのため，要求が正しいもので

あるかを確認する必要がある．

参考文献

[1] Nishtala, R., Fugal, H., Grimm, S., Kwiatkowski, M., Lee, H., Li, H. C., Mcelroy, R., Paleczny, M., Peek, D., Saab, P., Stafford, D., Tung, T. and Venkataramani, V.: Scaling Memcache at Facebook, The 10th USENIX Symposium on Networked Systems Design and Implementation (NSDI ' 13), USENIX Association, pp. 385–398 (2013).

[2] Amazon Web Services: Amazon ElastiCache, (online), available from ⟨https://aws.amazon.com/jp/elasticache/⟩ (accessed 2026-10-22).

[3] Chen, J., Chen, L., Wang, S., Zhu, G., Sun, Y., Liu, H. and Li, F.: HotRing: A Hotspot-Aware In-Memory Key-Value Store, The 18th USENIX Conference on File and Storage Technologies (FAST' 20), USENIX Association, pp. 239–252 (2020).

[4] Yang, J., Yue, Y. and Rashmi, K. V.: A large scale analysis of hundreds of in-memory cache clusters at Twitter, *The 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI'20)*, USENIX Association, pp. 191–208 (2020).

[5] National Institute of Standards and Technology: CVE-2025-49844 Detail, National Vulnerability Database (NVD) (online), available from ⟨https://nvd.nist.gov/vuln/detail/cve-2025-49844⟩ (accessed 2026-10-20).

[6] Redis: Redis, (online), available from ⟨https://redis.io/⟩ (accessed 2026-10-22).

[7] Hazan, Y.: Lua Use-After-Free may lead to remote code execution, (online), available from ⟨https://github.com/redis/redis/security/advisories/GHSA-4789-qfc9-5f9q⟩ (accessed 2026-10-20).

[8] Goel, A., Chopra, B., Gerea, C., Mátáni, D., Metzler, J., Haq, F. U. and Wiener, J. L.: Fast database restarts at Facebook, *the 2024 ACM SIGMOD International Conference on Management of Data (SIGMOD '14)*, Association for Computing Machinery, pp. 541–549 (2014).

[9] Filardo, N. W., Gutstein, B. F., Woodruff, J., Ainsworth, S., Paul-Trifu, L., Davis, B., Xia, H., Napierala, E. T., Richardson, A., Baldwin, J., Chisnall, D., Clarke, J., Gudka, K., Joannou, A., Markettos, A. T., Mazzinghi, A., Norton, R. M., Roe, M., Sewell, P., Son, S., Jones, T. M., Moore, S. W., Neumann, P. G. and Watson, R. N. M.: Cornucopia: Temporal Safety for CHERI Heaps, *2020 IEEE Symposium on Security and Privacy (S&P'20)*, IEEE (2020).

[10] Intel Corporation: Intel(R) 64 and IA-32 Architectures Software Developer's Manual, (online), available from ⟨https://www.intel.com/content/www/us/en/developer/technical-library/overview.html⟩ (accessed 2026-10-20).

[11] Gao, F., Wang, L. and Li, X.: BovInspector: Automatic inspection and repair of buffer overflow vulnerabilities, *The 31st IEEE/ACM International Conference on Automated Software Engineering (ASE'16)*, Association for Computing Machinery, Inc, pp. 786–791 (2016).

[12] Wickman, B., Hu, H., Yun, I., Jang, D., Lim, J., Kashyap, S. and Kim, T.: Preventing Use-After-Free Attacks with Fast Forward Allocation, *The Proceedings of the 30th USENIX Security Symposium (USENIX Security'21)*, USENIX Association, pp. 2453–2470 (2021).

[13] The MITRE Corporation: 2025 CWE Top 25 Most Dangerous Software Weaknesses, Common Weakness Enumeration (CWE) (online), available from ⟨https://cwe.mitre.org/top25/archive/2025/2025_cwe_top25.html⟩ (accessed 2026-10-20).

[14] Miller, M.: Trends, challenges, and strategic shifts in the software vulnerability mitigation landscape (2019).

[15] The MITRE Corporation: CVE-2021-32761, CVE (accessed 2026-10-19).

[16] Redis: BITFIELD, (online), available from ⟨https://redis.io/docs/latest/commands/bitfield/⟩ (accessed 2026-10-20).

[17] National Institute of Standards and Technology: CVE-2014-0160 Detail, National Vulnerability Database (NVD) (online), available from ⟨https://nvd.nist.gov/vuln/detail/cve-2014-0160⟩ (accessed 2026-10-20).

[18] Durumeric, Z., Kasten, J., Adrian, D., Halderman, J. A., Bailey, M., Li, F., Weaver, N., Amann, J., Beekman, J., Payer, M. and Paxson, V.: The Matter of Heartbleed, *The ACM SIGCOMM Internet Measurement Conference (IMC'14)*, pp. 475–488 (2014).

[19] Watson, R. N. M., Moore, S. W., Sewell, P. and Neumann, P. G.: An Introduction to CHERI, Technical report, The University of Cambridge Computer Laboratory (2019).

[20] Watson, R. N. M., Richardson, A., Davis, B., Baldwin, J., Chisnall, D., Clarke, J., Filardo, N., Moore, S. W., Napierala, E., Sewell, P. and Neumann, P. G.: CHERI C/C++ Programming Guide, Technical report, The University of Cambridge Computer Laboratory (2020).

[21] Lefeuvre, H., Badoiu, V. A., Jung, A., Teodorescu, S. L., Rauch, S., Huici, F., Raiciu, C. and Olivier, P.: FlexOS: Towards Flexible OS Isolation, *The 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'22)*, Association for Computing Machinery, pp. 467–482 (2022).

[22] Edge, J.: Rust for Linux redux, LWN.net (online), available from ⟨https://lwn.net/Articles/862018/⟩ (accessed 2026-10-21).

[23] Sartakov, V. A., Vilanova, L. and Pietzuch, P.: CubicleOS: A library OS with software componentisation for practical isolation, *The 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'21)*, Association for Computing Machinery, pp. 546–558 (2021).

[24] Kuenzer, S., Badoiu, V. A., Lefeuvre, H., Santhanam, S., Jung, A., Gain, G., Soldani, C., Lupu, C., Teodorescu, S., Raducanu, C., Banu, C., Mathy, L., Deaconescu, R., Raiciu, C. and Huici, F.: Unikraft: Fast, specialized unikernels the easy way, *The 16th European Conference on Computer Systems (EuroSys'21)*, Association for Computing Machinery, Inc, pp. 376–394 (2021).

[25] Vahldiek-Oberwagner, A., Elnikety, E., Duarte, N. O., Sammler, M., Druschel, P. and Garg, D.: ERIM: Secure, Efficient In-process Isolation with Protection Keys (MPK), *The 28th USENIX Security Symposium (USENIX Security'19)*, USENIX Association, pp. 1221–238 (2019).

[26] Litton, J., Vahldiek-Oberwagner, A., Elnikety, E., Garg, D., Bhattacharjee, B. and Druschel, P.: Light-Weight Contexts: An OS Abstraction for Safety and Performance, *The 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI'16)*,

ⓒ2026 Information Processing Society of Japan

USENIX Association, pp. 49–64 (2016).

[27] Wahbe, R., Lucco, S., Anderson, T. E. and Graham, S. L.: Efficient Software-Based Fault Isolation, *The fourteenth ACM symposium on Operating systems principles (SOSP'93)*, Association for Computing Machinery, Inc, pp. 203–216 (1993).

[28] Sehr, D., Muth, R., Biffle, C., Khimenko, V., Pasko, E., Schimpf, K., Yee, B. and Chen, B.: Adapting Software Fault Isolation to Contemporary CPU Architectures, *The 19th USENIX conference on Security (USENIX Security'10)*, USENIX Association (2010).

[29] Kim, J., Park, J., Lee, Y., Song, C., Kim, T. and Lee, B.: PeTAL: Ensuring Access Control Integrity against Data-only Attacks on Linux, *The 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS'24)*, Association for Computing Machinery, Inc, pp. 2919–2933 (2024).

[30] Edge, J.: Control-flow integrity for the kernel, LWN.net (online), available from ⟨https://lwn.net/Articles/810077/⟩ (accessed 2026-10-21).

[31] Android: Kernel control flow integrity, Android Open Source Project (online), available from ⟨https://source.android.com/docs/security/test/kcfi⟩ (accessed 2026-10-21).

[32] Microsoft: Control Flow Guard for platform security, Microsoft Learn (online), available from ⟨https://learn.microsoft.com/en-us/windows/win32/secbp/control-flow-guard⟩ (accessed 2026-10-21).

[33] Song, C., Lee, B., Lu, K., Harris, W., Kim, T. and Lee, W.: Enforcing Kernel Security Invariants with Data Flow Integrity, *The 23rd Annual Network and Distributed System Security Symposium (NDSS'16)*, The Internet Society (2016).

[34] Castro, M., Costa, M. and Harris, T.: Securing software by enforcing data-flow integrity, *The 7th USENIX Symposium on Operating Systems Design and Implementation (OSDI'06)*, USENIX Association, pp. 147–160 (2006).

[35] Edge, J.: Kernel address space layout randomization, LWN.net (online), available from ⟨https://lwn.net/Articles/569635/⟩ (accessed 2026-10-21).

[36] Arm: Memory Tagging Extension, (online), available from ⟨https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Arm_Memory_Tagging_Extension_Whitepaper.pdf⟩ (accessed 2026-10-21).

[37] Qualcomm: Pointer Authentication on ARMv8.3, (online), available from ⟨https://www.qualcomm.com/content/dam/qcomm-martech/dm-assets/documents/pointer-auth-v7.pdf⟩ (accessed 2026-10-21).

[38] google: leveldb, (online), available from ⟨https://github.com/google/leveldb⟩ (accessed 2026-10-21).

[39] Yu, J. Z., Watt, C., Badole, A., Carlson, T. E. and Saxena, P.: Capstone: A Capability-based Foundation for Trustless Secure Memory Access, *The 32nd USENIX Security Symposium (USENIX Security'23)*, USENIX Association, pp. 787–804 (2023).

[40] Arm: TrustZone technology, (online), available from ⟨https://support.arm.com/?tab=compute-ip⟩ (accessed 2026-10-21).

[41] Intel Corporation: Intel(R) Memory Protection Extensions Enabling Guide, (online), available from ⟨https://www.intel.com/content/www/us/en/developer/articles/guide/intel-memory-protection-extensions-enabling-guide.html⟩ (accessed 2026-10-21).

[42] Watson, R. N., Woodruff, J., Neumann, P. G., Moore, S. W., Anderson, J., Chisnall, D., Dave, N., Davis, B., Gudka, K., Laurie, B., Murdoch, S. J., Norton, R., Roe, M., Son, S. and Vadera, M.: CHERI: A hybrid capability-system architecture for scalable software compartmentalization, *2015 IEEE Symposium on Security and Privacy (S&P'15)*, Vol. 2015-July, Institute of Electrical and Electronics Engineers Inc., pp. 20–37 (2015).

[43] Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D. and Wood, D. A.: The gem5 Simulator, *ACM SIGARCH Computer Architecture News*, Vol. 39, p. 1 (online), DOI: 10.1145/2024716.2024718 (2011).

[44] Redis: Redis, (online), available from ⟨https://github.com/redis/redis⟩ (accessed 2026-7-23).

[45] Singer, J.: Capability misuse, (online), available from ⟨https://capabilitiesforcoders.com/compiler_errors/capability_misuse.html⟩ (accessed 2026-10-20).

[46] CTSRD-CHERI: cheribuild, (online), available from ⟨https://github.com/CTSRD-CHERI/cheribuild⟩ (accessed 2026-5-4).

ⓒ2026 Information Processing Society of Japan