# 削除済みVSSスナップショットの復元

株式会社インターネットイニシアティブ
小林 稔

# Who am I

- 小林　稔
- 株式会社インターネットイニシアティブ
  セキュリティ本部　セキュリティ情報統括室に所属
  - 社内外のインシデントレスポンスに従事
- 外部活動
  - *2015*年*8*月～*2018*年*1*月 社会保障審議会年金事業管理部会運営担当参与
  - *Mauritius 2016 FIRST Technical Colloquium* スピーカーおよびトレーニング講師
  - *2017*年セキュリティキャンプ全国大会講師
- Twitter : @unkn0wnbit

# Agenda

- 本発表の概要とモチベーション、目標
- VSSスナップショットのデータ構造
- VSSスナップショットの仕組み
- 主なスナップショットパーサ
- 削除済みVSSスナップショットアクセス手法の検討
- 作成したツールの概要と復元テスト
- Demo
- まとめ

# 概要

- 本発表はVolume Shadow Copy Service (VSS)に関する調査研究である。
- VSSはWindowsに標準搭載されているバックアップ関連機能で、NTFSボリュームのVSSスナップショット（以下、スナップショット）を作成することができる。
- スナップショットを参照すると、スナップショット作成時のデータ、つまり過去のデータにアクセスできる。そのため、攻撃の痕跡を発見できる場合があり、インシデントレスポンスにおいて重要な役割を果たす。しかし、容量の上限によって自然に古いものが消える、もしくは攻撃者やマルウェアによって削除されてしまう場合がある。
- 本発表では、削除されたスナップショットにアクセスする手法の検討とそれを基に実装したツールの実験結果を解説する。

# インシデントレスポンス時のVSSスナップショットの活用方法

- 時間の経過や攻撃者により削除されてしまったマルウェアや攻撃者の痕跡を復元し、解析することで、より深く解析を行う
  - 攻撃者が使用したツール
  - 攻撃者が一時的に作成したアーカイブファイル
  - 現在のイベントログから削除されてしまったログ
- ランサムウェアに暗号化されてしまったファイルの復元

# モチベーション (1)

- スナップショットは重要なアーティファクトであるが、削除されたスナップショットにWindowsからアクセスする方法は存在しない。
- X-waysなど一部のツールは特定の条件下で削除済みのスナップショットにアクセスできることをCDI山崎氏が確認している。
  - *http://www.kazamiya.net/DeletedSC*
- このことから、VSS関連のファイルを復元できれば、他のツールでもデータを復元できる可能性がある。これが第1のモチベーションである。

# モチベーション (2)

- 削除済みのスナップショットの一部にアクセスする手段の一つとして、カービングが存在する。しかし、この手法には致命的な欠陥が存在する。
- カービングはシグネチャを利用して、連続した領域をファイルやレコードとして復元する。しかし、スナップショットは16KBのデータブロック単位でバックアップが行われる。そのため、カービングを行っても最大で16KBのファイルやレコードしかカービングすることができない。また、ファイル生成日時などのメタ情報も復元できない。
- スナップショットのデータにアクセスする際には、現在のNTFSボリュームとスナップショット内のバックアップデータを正しく組み合わせる必要がある。したがって、NTFSボリュームをパースするカービングツールを使用しても対象ファイル全体をカービングすることはできない。
- このように、削除されたスナップショットのアクセスには専用のツールが必要となるが、誰でも自由に使えるソフトウェアがなかった。これが第2のモチベーションである。

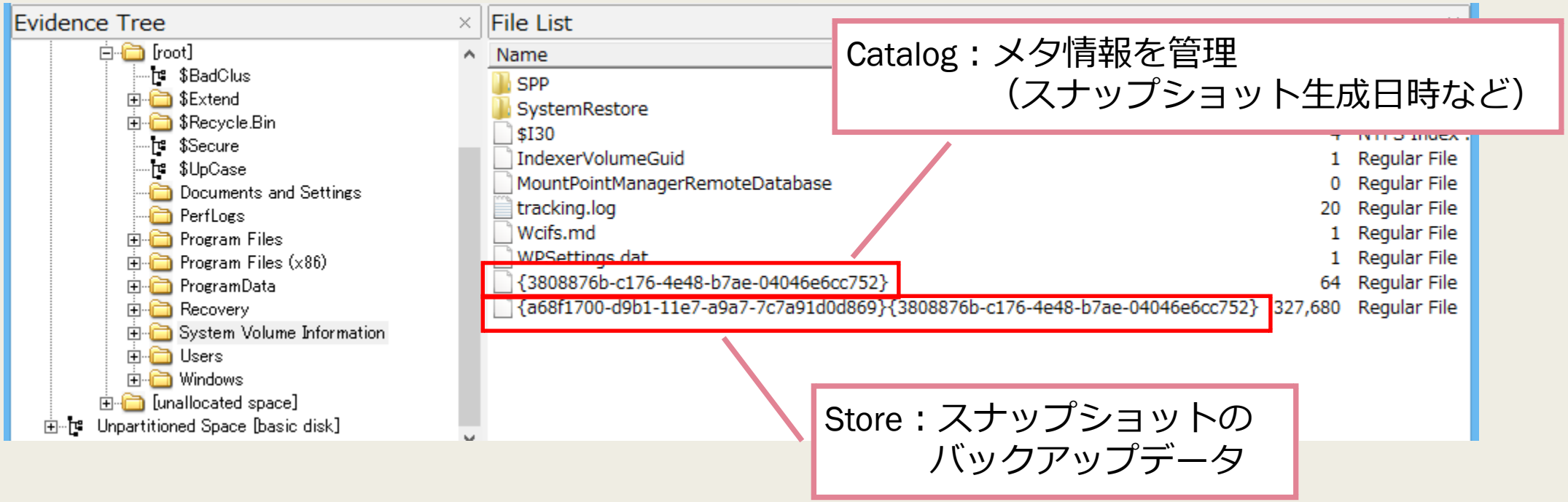
# 目標

- 以下のような状況で削除されたスナップショットからファイルを復元するツールを作成する。
  - スナップショットの容量オーバーで自動的に削除されたスナップショット
  - 攻撃者やランサムウェア等に削除されたスナップショット

# VSSスナップショットのデータ構造

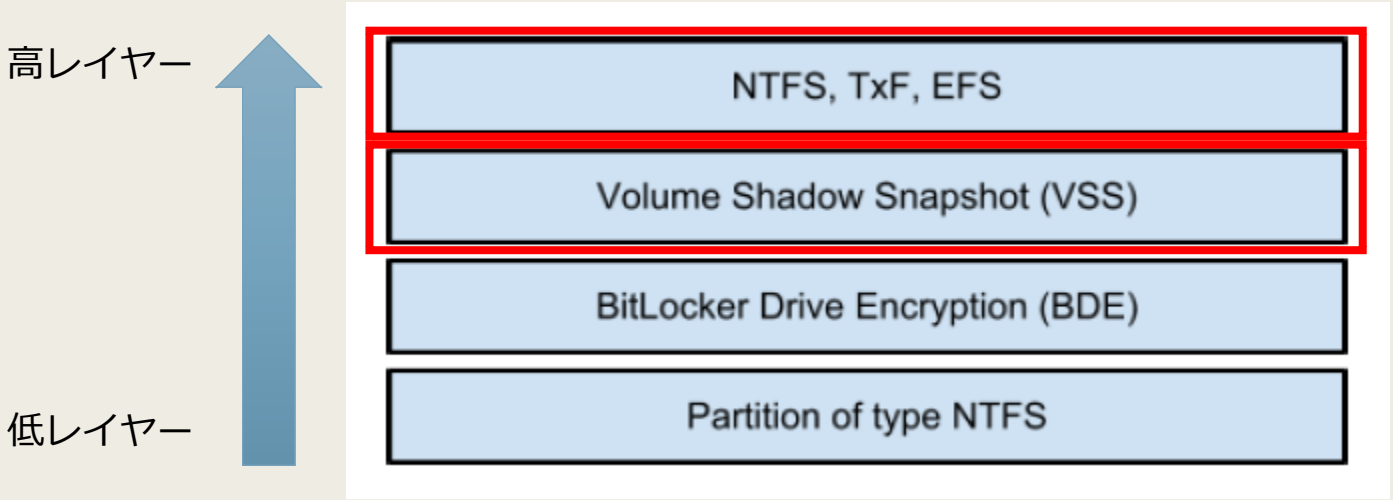
# VSSスナップショットのファイル

- VSSスナップショットの管理データはルートフォルダ直下のSystem Volume Informationに保存される



# VSSシステムのレイヤー

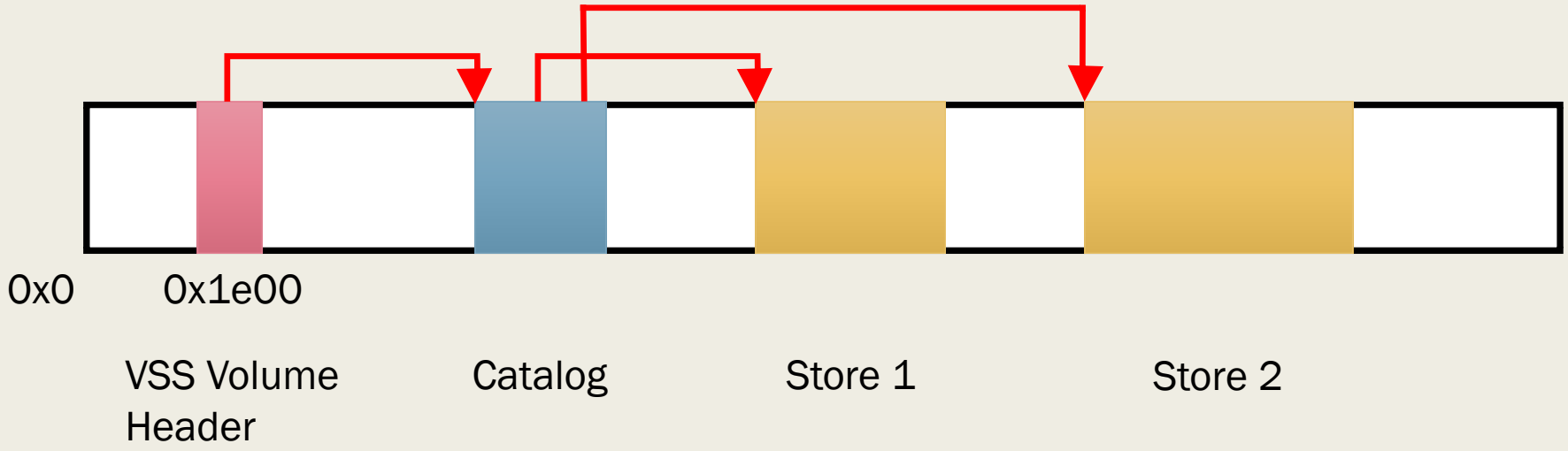
- VSSスナップショットの管理データはファイルとしても管理されているが、VSSはNTFSより下のレイヤーで動作しているため、VSSがスナップショットのデータを参照する際、NTFSをパースするのではなくオフセットをたどる。




https://github.com/libyal/documentation/blob/master/Paper%20-%20Windowless%20Shadow%20Snapshots.pdf

# VSSスナップショットへのアクセス

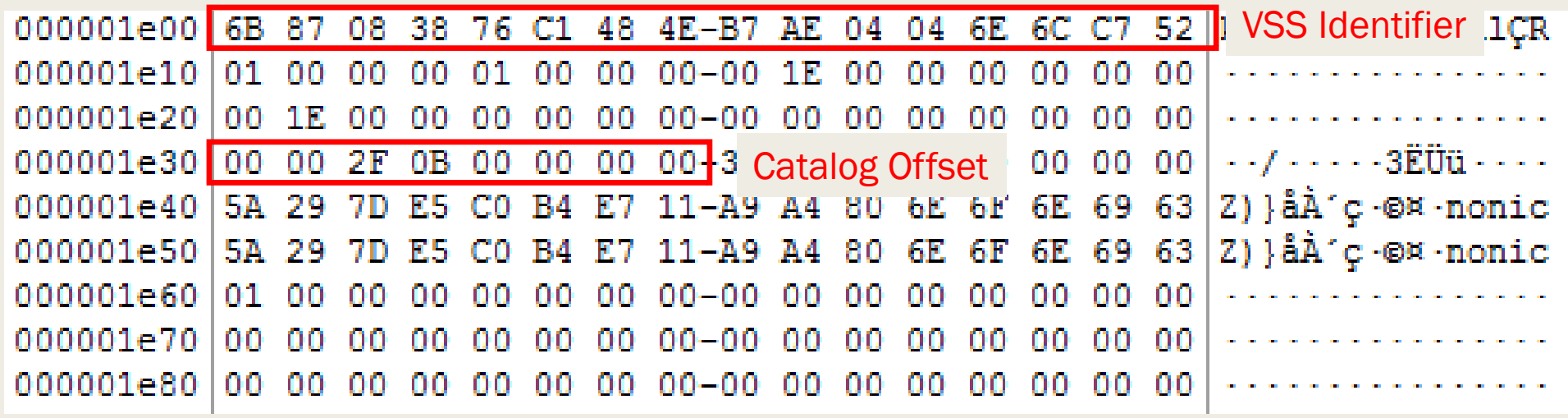
- オフセットをたどることでVSSスナップショットにアクセス



# VSS Volume Header (1)

- NTFSボリュームの先頭から0x1e00に保存される情報
- VSS Identifier
  - *VSS*が有効な場合にセットされる
  - *Catalog*や*Store*のデータブロックにもシグネチャとしてセットされる
- Catalog Offset
  - *Catalog*が保存されているボリュームの先頭からのオフセット
  - スナップショットが*0*個の場合、*0x0*がセットされる

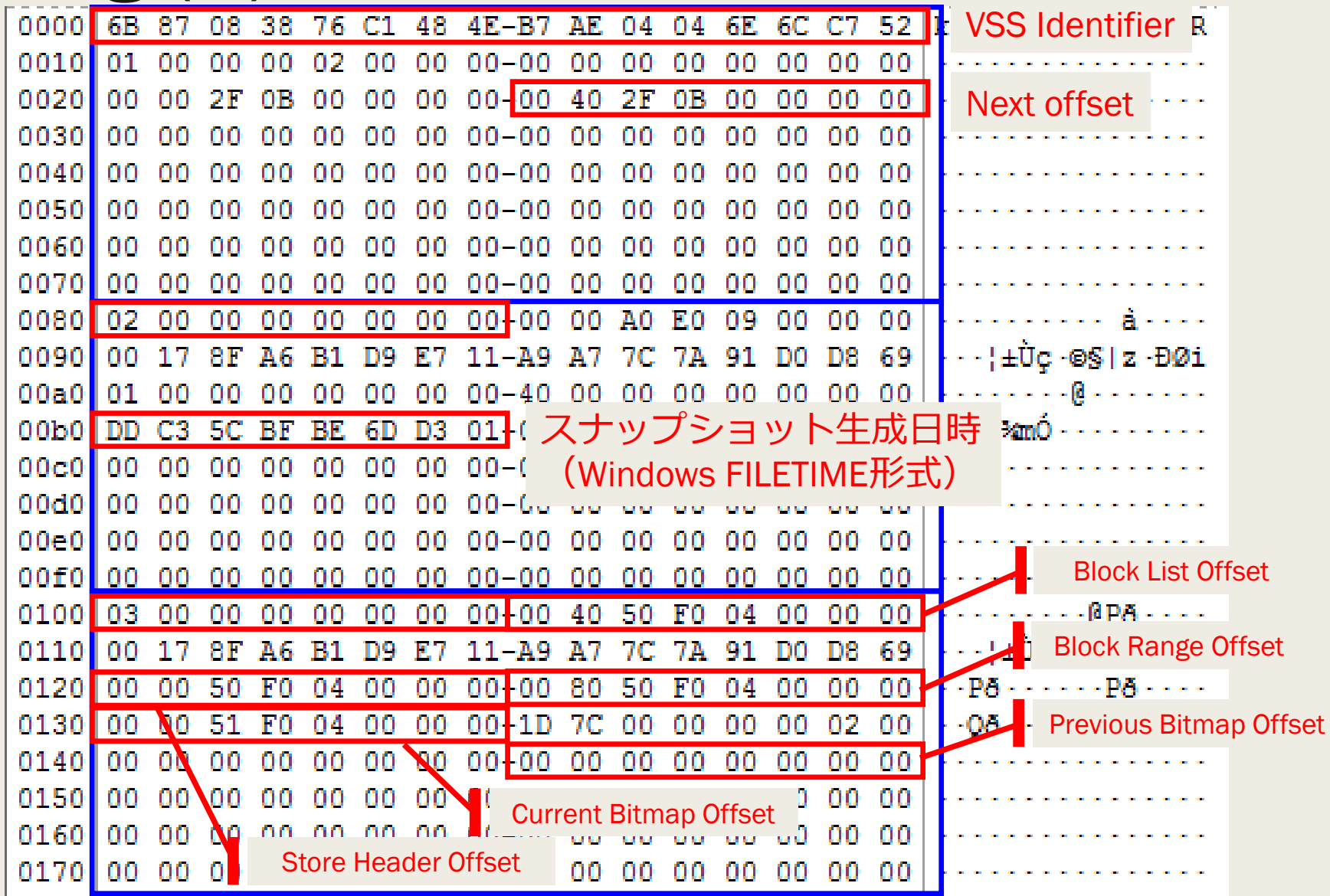
# VSS Volume Header (2)



# Catalog (1)

- Catalog Block Header
  - *VSS Identifier (Signature)*
  - *Next offset*
    - Catalogは初期状態で4データブロック確保されるため、オフセットがセットされる
- Catalog Entry
  - *Catalog Entry Type 0x02と0x03の2つ一組で1つのスナップショットを管理している*
  - *Catalog Entry Type 0x02*
    - スナップショット生成日時など
  - *Catalog Entry Type 0x03*
    - Store Header Offset, Store Block List Offset, Store Block Range Offset, Store Current Bitmap Offset, Store Previous Bitmap Offset, など

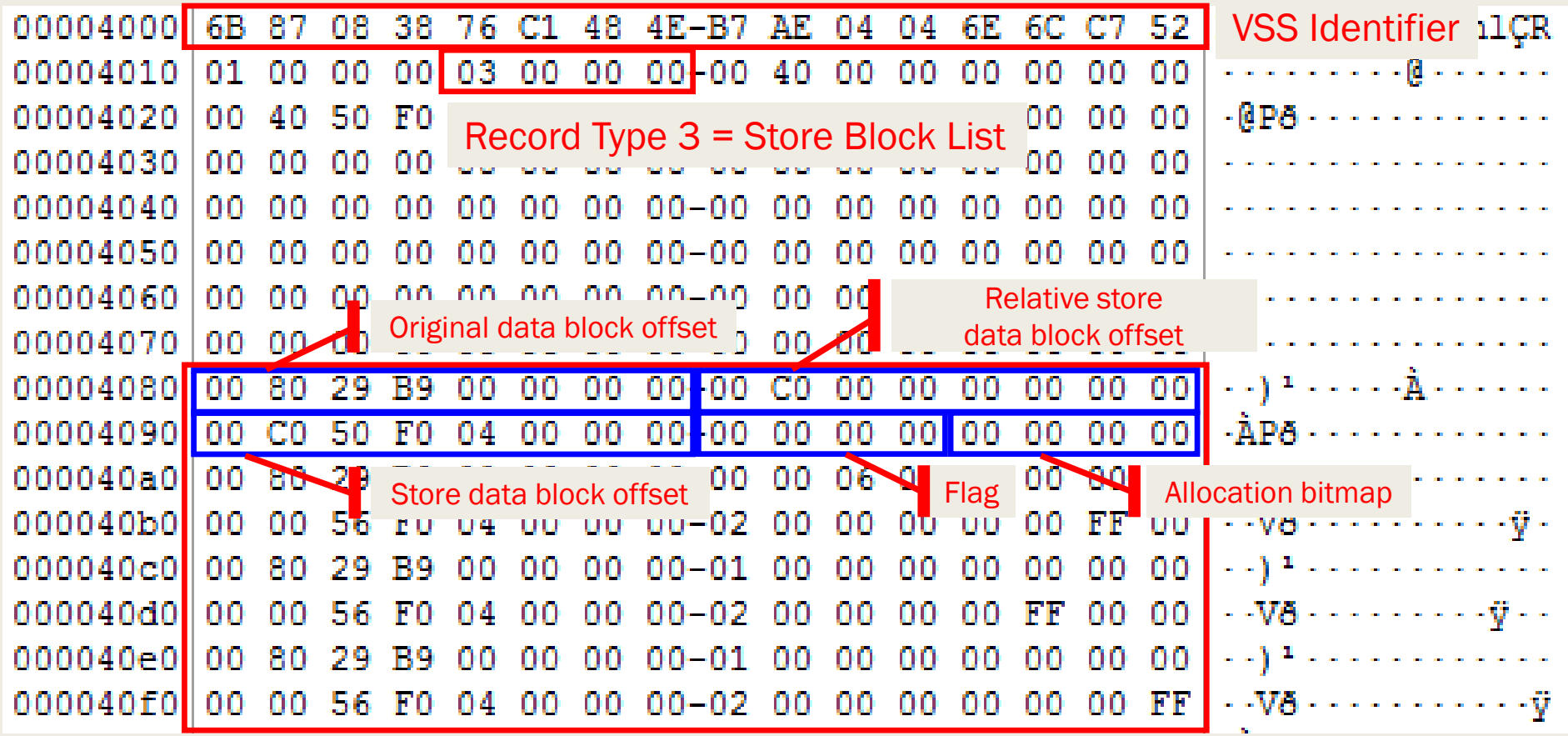
# Catalog (2)


Catalog Block Header
VSS Identifier
Next offset
Entry Type 0x02
スナップショット生成日時
（Windows FILETIME形式）
Entry Type 0x03
Block List Offset
Block Range Offset
Previous Bitmap Offset
Current Bitmap Offset
Store Header Offset

# Store (1)

- Store Block Header
  - *4*種類の*Record Type*の*Store Block*で*1*つの*Store*は管理されている
- Store Header (Store Information) : Record Type 4
  - スナップショット*GUID*、属性フラグ、マシン名など
- Store Block List : Record Type 3
  - データブロックが記録されていたオリジナルのオフセットとバックアップ先のオフセットのテーブル
- Store Block Range : Record Type 5
  - *Store*自身が使用しているファイル領域のオフセットとレンジのリスト
- Store Current Bitmap / Store Previous Bitmap : Record Type 6
  - ボリュームのデータブロックの利用状況を表すビットマップ
- Store Data Block
  - バックアップされたデータブロック

# Store (2) - Store Block Listの例

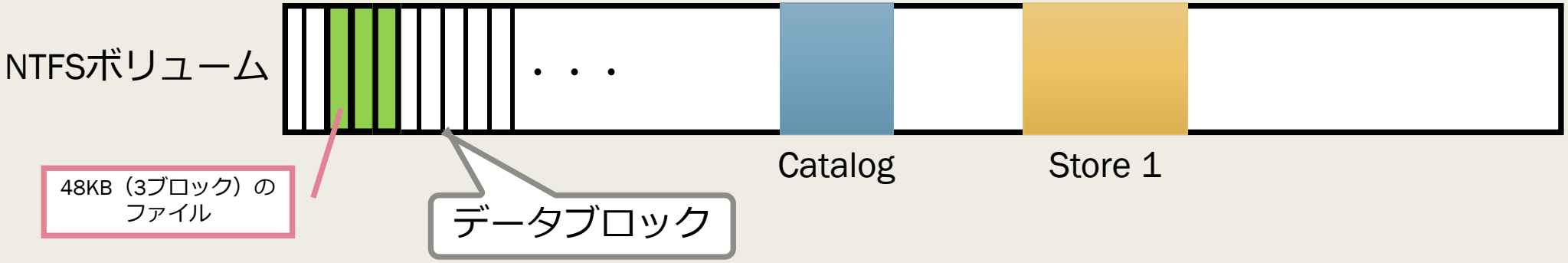


# VSSスナップショットの仕組み

# VSSスナップショットのデータ保存

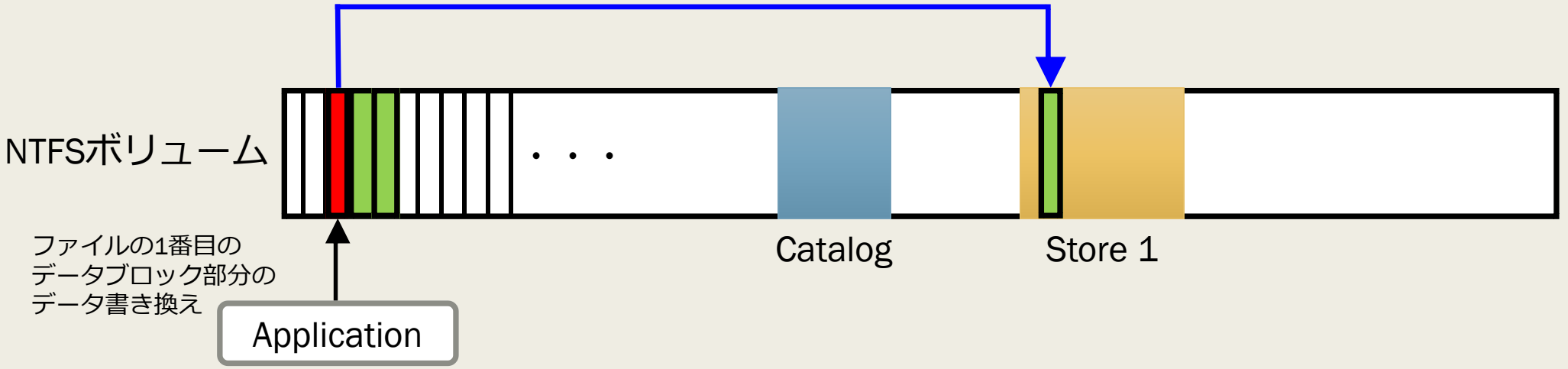
# VSSスナップショットのデータ保存(1)

- スナップショットを作成するとスナップショット作成日時などのメタ情報を保存するCatalogとバックアップデータを保存するStoreが用意される。
- NTFSボリュームは16KBごとのデータブロックと呼ばれる単位で管理される。
- 例として、3つのデータブロックを使用するファイルのスナップショットについて考えてみる。



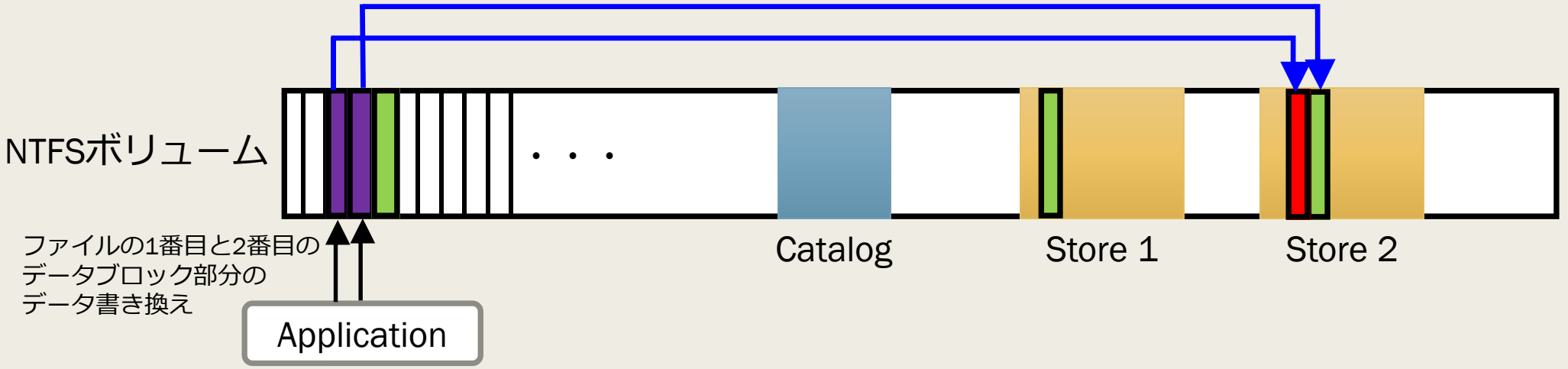
# VSSスナップショットのデータ保存 (2)

- ファイルの保存などでデータ変更が発生したデータブロックはデータが書き込まれる前にStoreにデータブロック単位でバックアップされる。



# VSSスナップショットのデータ保存(3)

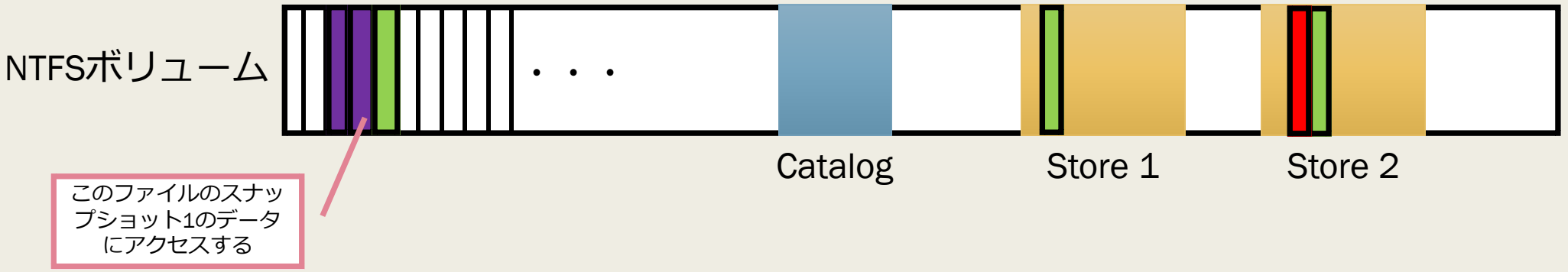
- 2つ目のスナップショットを作成するとCatalogへのメタ情報の追加と2つ目のStoreが用意され、変更が発生したデータブロックはStore 2にバックアップされる。
- 3つ目以降のスナップショットも同様。



# VSSスナップショットのデータアクセス

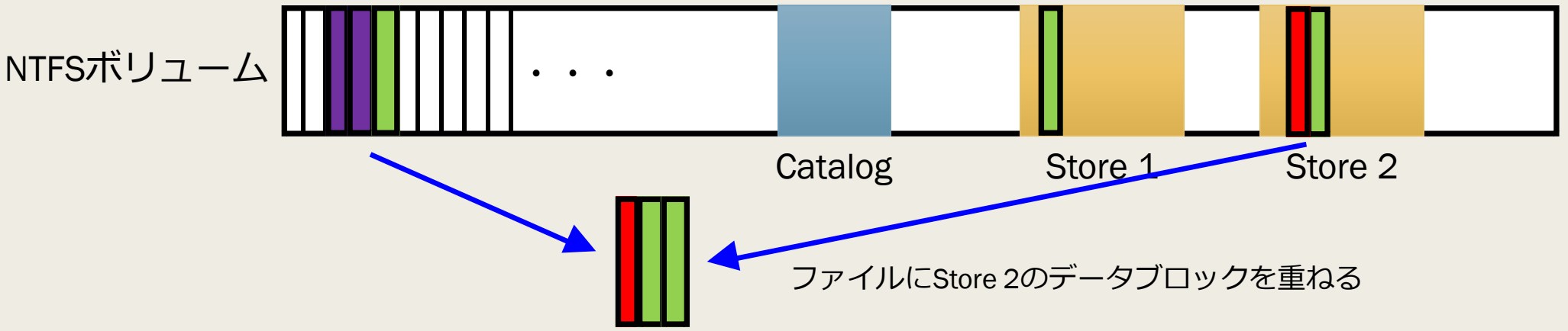
# VSSスナップショットのデータアクセス(1)

- スナップショットのデータにアクセスする際、Storeに保存されているデータブロックと現在のNTFSボリュームを組み合わせて、スナップショット作成時のデータブロックを再現する。
- 例として、スナップショット1作成時のファイルにアクセスする場合を考える。



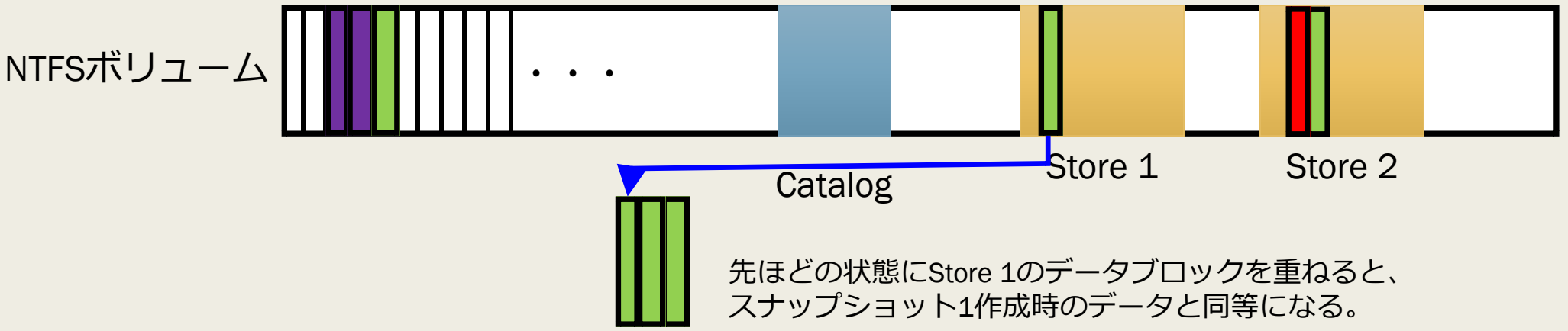
# VSSスナップショットのデータアクセス (2)

- Store2に保存されているデータブロックをファイルに重ね合わせる。



# VSSスナップショットのデータアクセス (3)

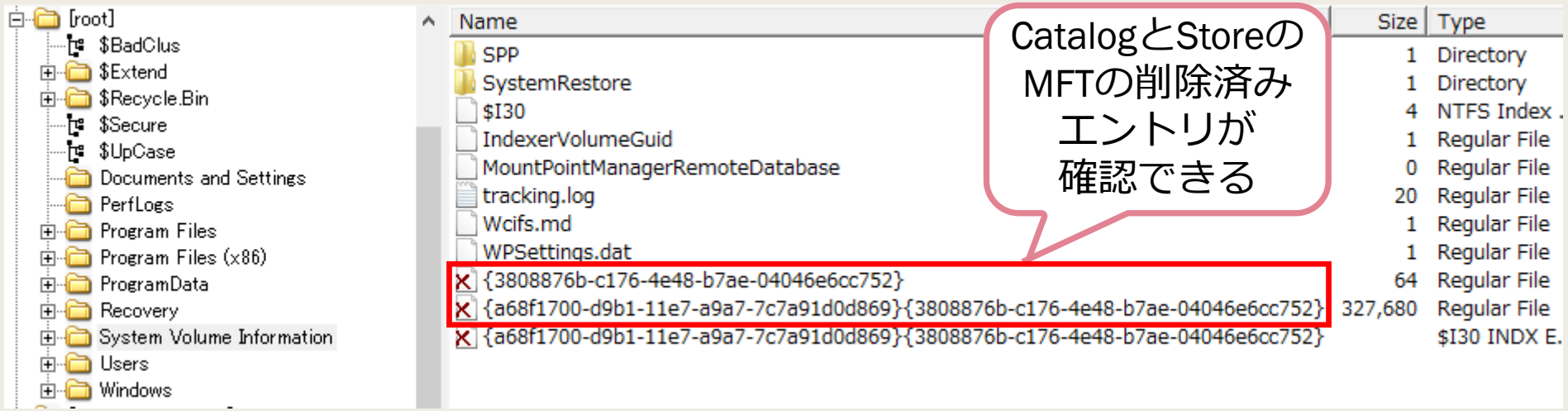
- Store1に保存されているデータブロックをファイルに重ね合わせる。
- このように、スナップショット内にバックアップされたデータブロックを新しい順にNTFSボリュームのデータブロックに重ねることで、スナップショット作成時のデータにアクセスすることができる。



# VSSスナップショットの削除

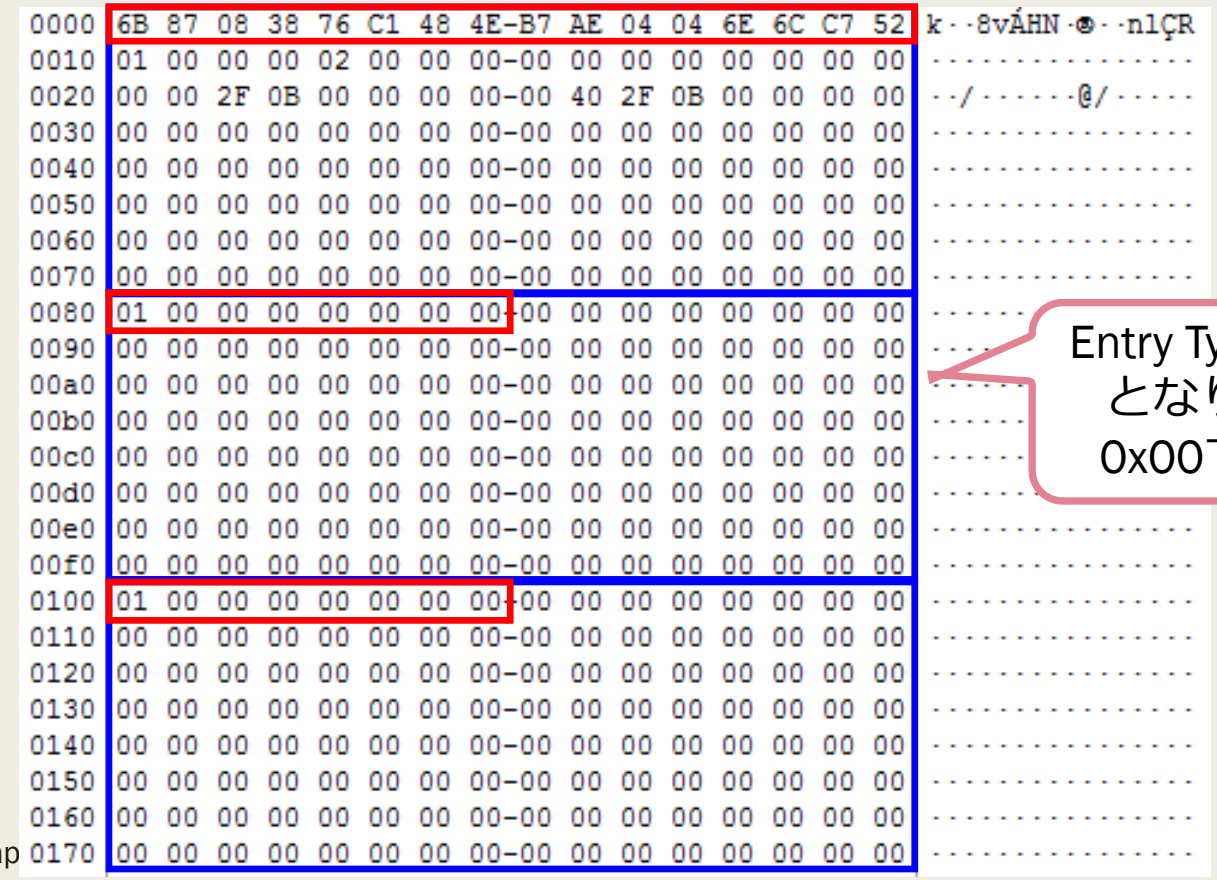
# スナップショット削除時の動作 (1)

- スナップショット削除コマンド
  - *vssadmin.exe delete shadows /all*
- スナップショット削除直後のCatalogとStoreの状態



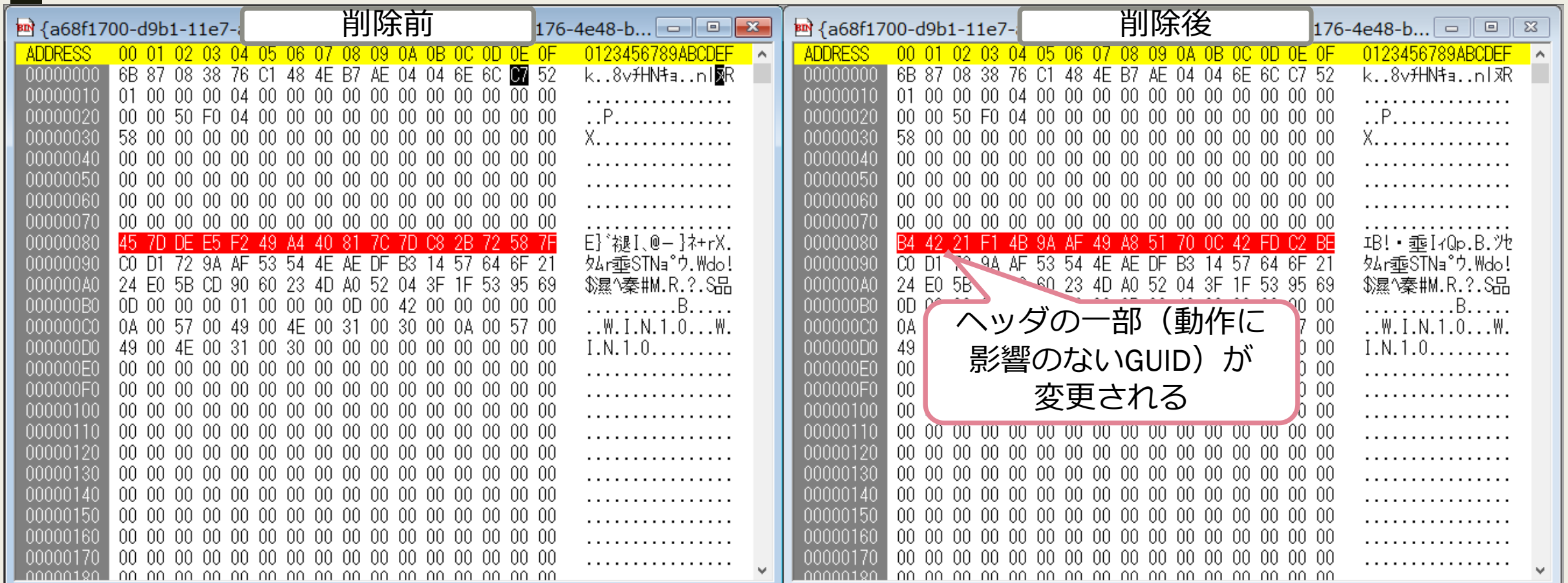
# スナップショット削除時の動作 (2)

- スナップショット削除直後のCatalog のファイル内容



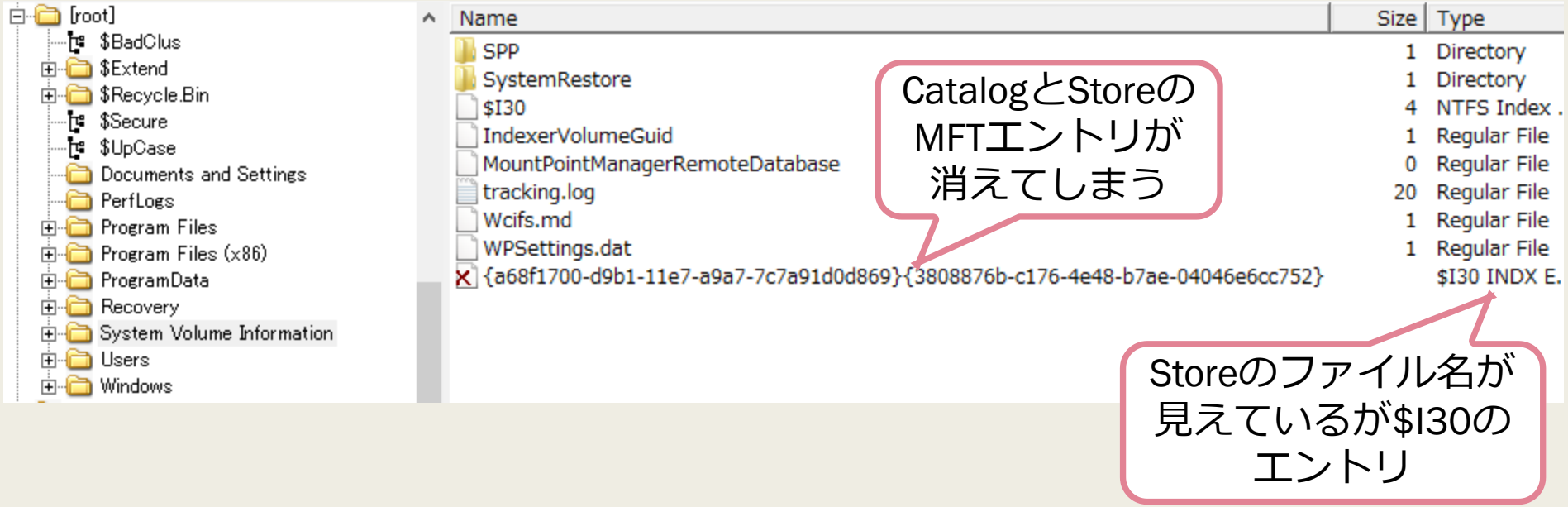
# スナップショット削除時の動作 (3)

- スナップショット削除直後のStore のファイル内容



# スナップショット削除時の動作 (4)

- スナップショット削除から数分経過後



# 主なVSSスナップショットパーサ

# 主なVSSスナップショットパーサ

- Forensic Tool Kit、X-ways Forensics、AXIOM、EnCase などの商用製品
- ShadowExplorer、ShadowKit などのフリーソフトウェア
- libvshadow などのオープンソースソフトウェア
- ほとんどのツールは削除されたスナップショットにアクセスできない
  - *X-Ways*は*MFT*に*Catalog*と*Store*の削除エントリが残っていればスナップショットにアクセス可能。
  - しかし、スナップショット削除後、数分経過するとこれらのエントリは*MFT*から削除されてしまうため、実用性は高くない。
- 今回は、libvshadow を作成するツールのベースとして採用した。

# なぜ、libvshadowなのか

- 商用製品がVSSスナップショットをうまく処理できない場合でも、libvshadowであれば正常に処理できることが多くある。
- ライブラリだけではなく、スナップショットをディスクイメージとして再現するvshadowmountというコマンドがあるため、他のディスクイメージを処理するツールと組み合わせやすい。
- Windowsの機能を使用せずにVSSスナップショットパーサを実装しており、オープンソースソフトウェアであるため、機能の拡張がしやすい。
- 誰でも使用することが可能。
- https://github.com/libyal/libvshadow

# 削除済みVSSスナップショットアクセス手法の検討

# 削除済みVSSスナップショットアクセス手法の検討 (1)

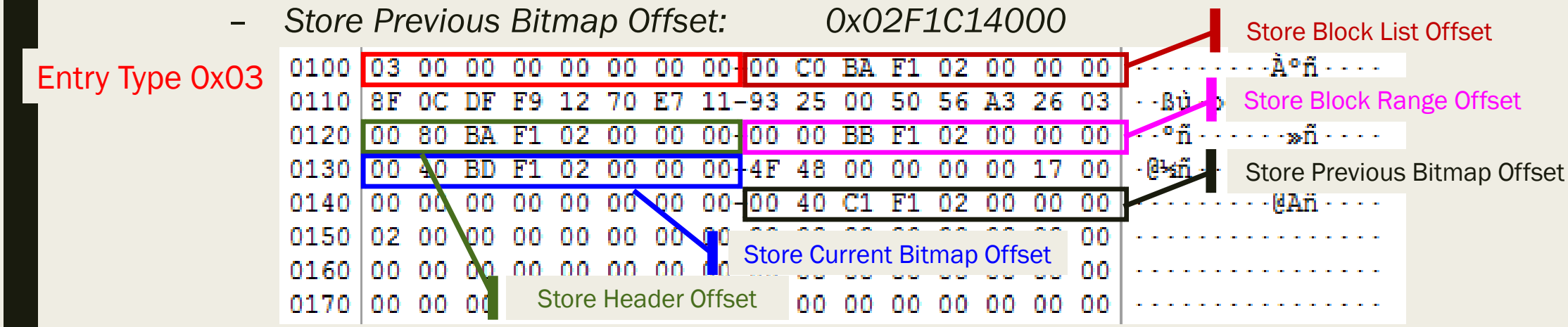
- vshadowmountはディスクイメージ内のCatalogとStoreを読み込んでスナップショットのデータにアクセスするため、CatalogとStoreを復元または再生成する必要がある。
- しかし、CatalogとStoreの復元には以下のような問題が存在する。
  1. *Store*については動作に影響しない一部のデータのみが書き換わるだけであるため、ディスクイメージからカービングしたデータを利用できることが期待できる。しかし、*Store*は*4*種類のストアブロックから構成されるため、カービングしたストアブロックを*1*つの*Store*してグルーピングする方法が必要となる。
  2. *Catalog*のデータは完全に失われてしまうため、カービングした*Store*などから再生成する必要がある。
  3. 複数の*Store*がカービングできた場合、*Store*が作成された順番が分からない。

# 削除済みVSSスナップショットアクセス手法の検討 (2)

- 問題点1
  - *Store*については動作に影響しない一部のデータのみが書き換わるだけであるため、ディスクイメージからカービングしたデータを利用できることが期待できる。しかし、*Store*は*4*種類のストアブロックから構成されるため、カービングしたストアブロックを*1*つの*Store*してグルーピングする方法が必要となる。
- 解決策1
  - 各種ストアブロックの*NTFS*ボリューム上での配置を調査して、グルーピングの方法を検討する。

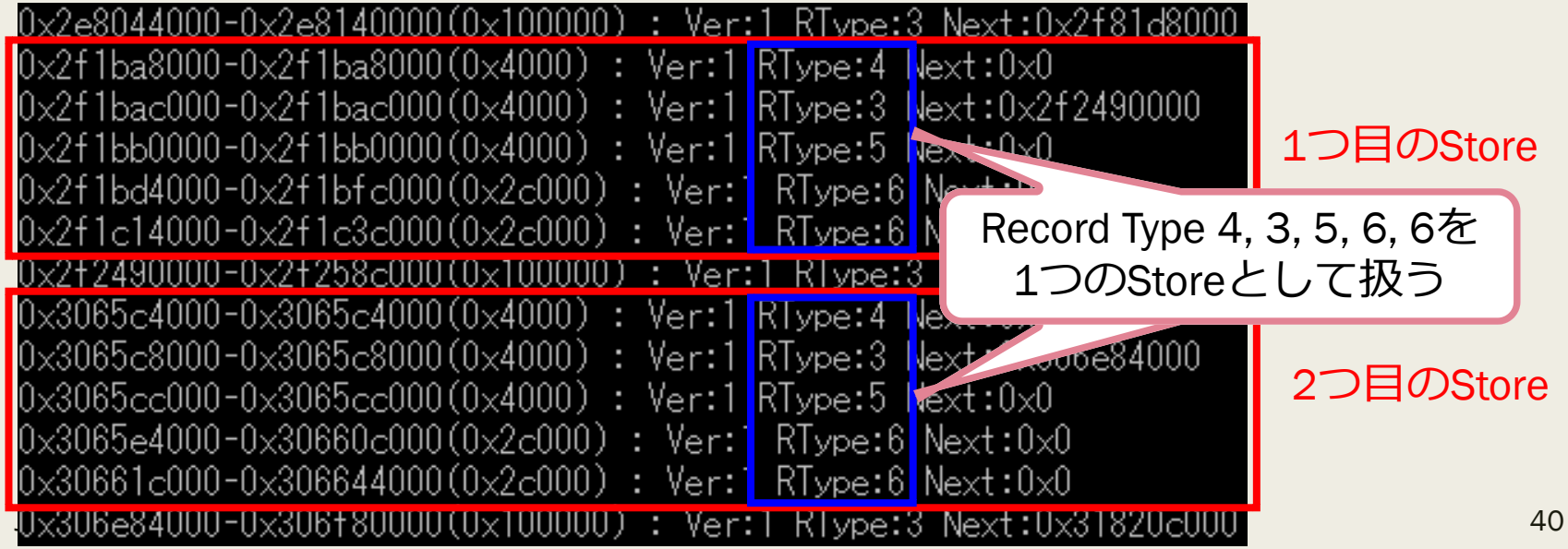
# 削除済みVSSスナップショットアクセス手法の検討 (3)

- CatalogのEntry Type 0x03に記録されている各種ストアブロックのオフセットを見るとある程度の範囲に収まっていることが分かる（NTFSボリュームの大きさによって変化する）。
  - *Store Header Offset:* *0x02F1BA8000*
  - *Store Block List Offset:* *0x02F1BAC000*
  - *Store Block Range Offset:* *0x02F1BB0000*
  - *Store Current Bitmap Offset:* *0x02F1BD4000*
  - *Store Previous Bitmap Offset:* *0x02F1C14000*



# 削除済みVSSスナップショットアクセス手法の検討 (4)

- 次に、ディスクイメージ内のストアブロックを検索するツールを作成し、他のStoreのストアブロックと入り組んだ配置になっていないか確認すると、Storeごとに離れて配置されていることが確認できる。
- また、StoreはRecord Type 4, 3, 5, 6, 6の順番でストアブロックが現れるため、これらを1つのStoreとしてカービングしても問題ないであろうと考えられる。



# 削除済みVSSスナップショットアクセス手法の検討 (5)

- 問題点2
  - *Catalog*のデータは完全に失われてしまうため、カービングした*Store*などから再生成する必要がある。
- 解決策2
  - *Catalog*に必要な主な情報は以下の通り。
    - スナップショット生成日時
    - Store Header Offsetなどの各オフセット
  - オフセットはカービングした*Store*から取得することができるが、スナップショット生成日時は完全に失われている。
  - *vshadowmount*はスナップショット生成日時の新しい順にスナップショットを並べ替えてスナップショットのデータにアクセスする。つまり、スナップショット間で前後関係が正しければ、生成日時はいつでも良い。
  - このことから、カービングを行った日時を基準にスナップショット生成日時をセットすることとした（問題点3と関連）。

# 削除済みVSSスナップショットアクセス手法の検討 (6)

- 問題点3
  - 複数の*Store*がカービングできた場合、*Store*が作成された順番が分からない。
    - スナップショットのデータを参照する際に、正しい順番でデータブロックを再構成しないと正しいデータを再現することができない。
- 解決策3
  - 新しい*Store*は*NTFS*ボリュームのオフセットが大きい位置に配置されると仮定し、*Catalog*を生成する際に、一番大きなオフセットの*Store*のスナップショット生成日時にカービングを行った日時をセットし、*1*つ前のスナップショット生成日時には、*1*時間前の日時をセットするようにした（以降、同様に*1*時間ずつ前の日時をセットする）。
  - しかし、実際には新しい*Store*が小さなオフセットに作成される場合もある。このような状況はツールで自動的に判断できないため、利用者がスナップショットを読み込む順番を簡単に入れ替えられるようなツールを別途作成した。

# 作成したツールの概要と復元テスト

# 作成したツール

- vss_carver.py （カービングツール）
  - ディスクイメージから*Store*をカービングする
  - カービングした*Store*から*Catalog*を生成する
  - ディスクイメージ内に*Catalog*があれば、その情報とカービングした情報をマージする（情報は*Catalog*を優先する）
- vss_catalog_manipulator.py （カタログを操作するツール ）
  - カタログエントリの順番の変更や削除などを行う
- 拡張版vshadowmount （libvshadow-20170902をベースに作成）
  - *vss_carver.py*で復元した*Catalog*と*Store*からスナップショットの情報を読み込むオプションを追加した

# vss_carver.pyの使い方

- -o / --offset : ディスクイメージ内のNTFSボリュームのオフセット
- --i / --image : ディスクイメージのファイルパス
- -c / --catalog : Catalog書き出し先のファイルパス
- -s / --store : Store書き出し先のファイルパス
- vss_carver.py -o 123456 -i y:¥image -c z:¥catalog -s z:¥store

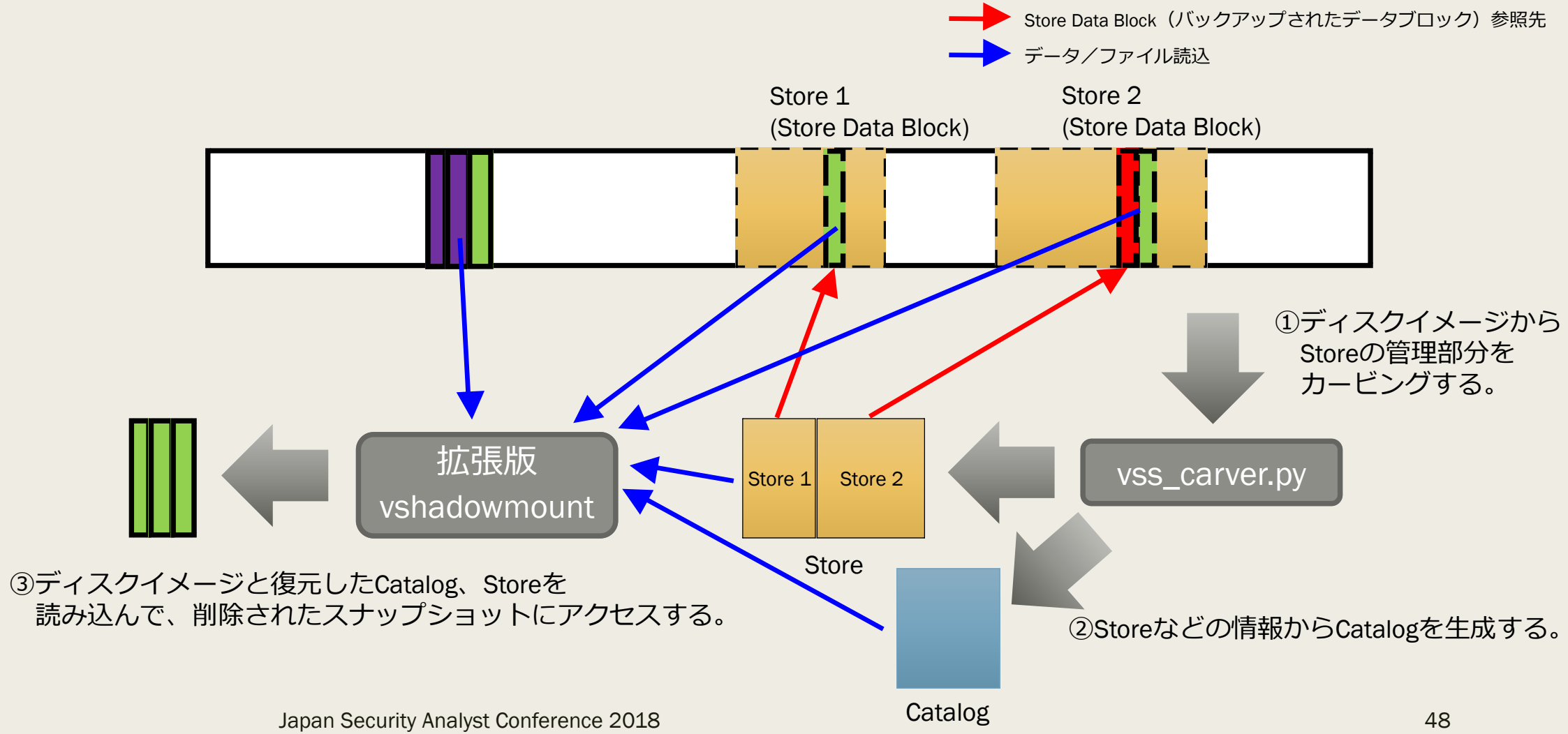
# vss_catalog_manipulator.pyの使い方

- list : Catalog内の情報を出力
  - *vss_catalog_manipulator.py list z:¥catalog*
- move : Catalogの5番のエントリを3番の位置に移動する
  - *vss_catalog_manipulator.py move z:¥catalog 5 3*
- remove : Catalogの2番のエントリを削除する
  - *vss_catalog_manipulator.py remove z:¥catalog 2*
- enable : Catalogの4番のエントリを有効化する
  - *vss_catalog_manipulator.py enable z:¥catalog 4*
- disable : Catalogの7番のエントリを無効化する
  - *vss_catalog_manipulator.py disable z:¥catalog 7*

# 拡張版vshadowmountの使い方

- 既存のオプションは変更せず、新たにオプションを追加した
- -c : Catalogの情報としてvss_carver.pyで生成したCatalogファイルを使用する
- -s : Storeの情報としてvss_carver.pyでカービングしたStoreファイルを使用する
- vshadowmount.exe -o 123456 -c z:¥catalog -s z:¥store y:¥image x:

# 作成したツールの概要


Store Data Block（バックアップされたデータブロック）参照先
データ／ファイル読込
Store 1
(Store Data Block)
Store 2
(Store Data Block)
①ディスクイメージから
Storeの管理部分を
カービングする。
vss_carver.py
Store 1
Store 2
Store
拡張版
vshadowmount
③ディスクイメージと復元したCatalog、Storeを
読み込んで、削除されたスナップショットにアクセスする。
②Storeなどの情報からCatalogを生成する。
Catalog

# ファイル復元テスト

- 以下の操作を行ったディスクイメージからファイルを復元できるか、複数のスナップショットパーサでテストを行った。
- 事前準備
  - *3KB, 5MB, 15MB* のファイルを*10*個ずつ保存し、スナップショットを作成後、各ファイルの先頭に*1*バイトのデータを追加してファイルを保存。
- テスト1
  - 全てのスナップショット削除後、*1*分以内のディスクイメージ（*MFT*に削除エントリが残っている状態）
- テスト2
  - 全てのファイルを削除し、別の*10*個の*5MB*のファイルのコピーと削除を*5*回繰り返す。
- テスト3
  - *Teslacrypt*を実行して、ファイルを暗号化させる。
    - 外部への通信ができない検証環境で実行する必要があったため、通信ができなくともファイルを暗号化するTeslacryptを使用した。

# ファイル復元テスト結果

| ソフトウェア | テスト1 | テスト2 | テスト3 | 備考 |
|---|---|---|---|---|
| 商用製品A (Ver. X) | ○ | × | × | MFTにStoreのエントリがある場合のみ復元可能 |
| 商用製品A (Ver. Z) | × | × | × | 上記と同じ製品だが新しいバージョンにも関わらず、エンバグしている可能性がある |
| 商用製品B | × | × | × | |
| フリーソフトウェアC | × | × | × | |
| libvshadow + vss_carver.py | ○ | ○ | ○ | |

○：すべてのファイルが復元できた
×：1つもファイルが復元できなかった

# DEMO

# Demo 1: 自動的に削除されたスナップショットの復元

- 1か月程度運用したWindows 7のディスクイメージ
- ディスクイメージ内のCatalogには3つのスナップショットのエントリが記録されているが、カービングを行うと、自動的に削除されたスナップショットのエントリを見つけることができる。
- この削除されたスナップショットから既存のスナップショットより古いデータを復元できる可能性がある。

# Demo 2: ランサムウェアに削除されたスナップショットの復元

- 被害者コンピュータ：Windows 10
  - スナップショットを作成。既存のファイルを編集・保存する。
  - *Teslacrypt*を実行後、暗号化が完了したことを確認して、*VMware*のスナップショットを作成する。
- 解析コンピュータ：Windows 7
  - 被害者コンピュータの*VMware*ディスクイメージマウントして、*vshadowinfo*でスナップショットがないことを確認する（*Teslacrypt*による削除）。
  - *vss_carver.py*がカービングした*Catalog*と*Store*を使って、拡張版*vshadowmount*で*VMware*ディスクイメージをマウントする。
  - 暗号化される前のファイルが復元できるか確認する。

# ScopeSnapshotsの確認

- VSSスナップショットには、もう一つ考慮しなければならない点がある。
- Windows 8以降、デフォルトでScopeSnapshotsが有効になっている。
- ScopeSnapshotsが有効になっていると、VSSスナップショットにはシステム関連のファイルのみが保存され、ユーザが作成したデータは保存されない。
- ScopeSnapshotsを無効化するには、「HKLM¥Software¥Microsoft¥Windows NT¥CurrentVersion¥SystemRestore」キーに「ScopeSnapshots」という名前でDWORD値「0」を設定し、OSを再起動する。
- 詳細は弊社発行の冊子IIR Vol.37を参照。
  - *https://www.iij.ad.jp/dev/report/iir/037.html*

# Future Work

- 拡張版vshadowmountのWindows以外のOSへの対応
- ストアブロックの一部が上書きされている場合の補完処理

# まとめ

- ディスクイメージからStoreとCatalogを復元し、これらのファイルを機能拡張したvshadowmaountで読み込むことで削除されたVSSスナップショットのデータにアクセスできるようになった。
- また、自然に削除されたスナップショットやランサムウェアによって削除されたスナップショットにも有効であることが確認できた。
- 作成したツールの公開先と公開予定時期
  - *https://github.com/mnrkbys/vss_carver*
  - 拡張版*vshadowmount*パッチ公開予定時期：*2018*年2月
    - バイナリは先行して公開中

# Q & A