Vol.2018-CSEC-82 No.49
Vol.2018-SPT-29 No.49
2018/7/26

ファイルサーバーにおけるHIDSの適用研究

宮脇 一晃[1,a)]　辰己 丈夫[2,b)]

概要：情報セキュリティを確保するために，現在行われている対応の主流は，ファイルサーバーに対して，「ファイル，フォルダに対してのアクセスコントロール (ACL) による読み書き可能範囲を制限することで情報流出抑止，削除，改ざん防止とする」というものである．だが，この対応では，正当なアクセス権を行使した改ざん，削除を検知できない．一般的なサーバーのシステム領域および Web コンテンツの改ざん検知では，ホスト型 IDS が使用されるが，数 TiB クラスのストレージを持つファイルサーバーに適用した場合，実用的な時間内に改ざん，削除を検知をすることは容易ではない．筆者らは，数 TiB クラスのストレージを持つファイルサーバーを用い，実用的な時間で，改ざん，削除を検知し，復元する方法の検討を行い，部分的な検証を行った．

キーワード：ファイルサーバー, セキュリティ, ホスト型 IDS

Kazuaki Miyawaki[1,a)]　Takeo Tatsumi[2,b)]

1. はじめに

企業において多数のユーザーが情報共有，保存を行う上で，ファイルサーバーの重要性が増している．しかし，昨今の情報セキュリティにおいて，ファイルサーバーへの脅威も以下のように増えている．

- 旧来のウィルス等 (マルウェア)
- ランサムウェア
- ユーザーによる情報漏洩
- ユーザーの故意，過失による削除，改ざん

旧来のウィルス等 (マルウェア) は主に PC に感染し，ユーザーに知られないように情報を窃取，ボット PC 化するものが多く見受けられたが，ここ数年流行しているランサムウェアはユーザーのファイルを暗号化ロックし復元するために金銭を要求するものとなっている．また，ファイルサーバーの情報セキュリティ侵害はマルウェアだけでは無く，ユーザーの故意，誤操作による，削除，改ざんもファイルサーバーへの脅威となる．

つぎに，サーバーを維持する上でのセキュリティ 3 原則の要求内容を示す．

機密性　許可されたユーザーのみアクセス可能

完全性　故意，過失による，削除，改ざんの検知，復旧

可用性　冗長化，バックアップ

セキュリティ 3 原則のうち機密性が有効な場合，マルウェアは感染した PC のユーザーがアクセスできる範囲でしかファイルに対して感染，暗号化ロックはできない．また，ユーザーの故意，過失による削除，改ざんもユーザーがアクセスできる範囲でしか行えない．

ファイルサーバーで頻繁にアクセスされる領域であれば，ファイル，フォルダの削除，改ざんに気がつくことがあるが，ユーザーがたまにしかアクセスしない低アクセス頻度のアーカイブ領域を削除，改ざんされた場合，削除，改ざんが発覚するまでに時間がかかる場合がある．

このため，可用性を担保するバックアップでは，バックアップ保管期間を過ぎて発覚すると，データは永久に無くなる．

一般的なバックアップでは通常ファイルサーバーの利用頻度の低い夜間に行うため，バックアップで戻ることのできる RPO(RecoveryPointObjective) は夜間に行った，バックアップの実施時刻となり，バックアップ後に作成したファイルについては，復元できない．

同じく，ファイルサーバーの故障対策，BCP として利用される，サーバー冗長 (拠点内/遠隔地同期) は一定間隔で同期されるため，削除，改ざんがあった場合の復元もととしては有効ではない．

[1] 放送大学 大学院 Open University Japan
[2] 放送大学 Open University Japan
[a)] 1418214556@campus.ouj.ac.jp
[b)] ttmtko@ouj.ac.jp

ファイルサーバーにおいてセキュリティ 3 原則の内，機密性，可用性は OS 標準機能と，ハードウェアで維持可能なため，比較的実施されていることが多いが，完全性については，OS 標準機能には存在しないため，完全性を維持する仕組みを導入する難易度が高いと考える．

完全性を維持することが難しい理由は，ファイルサーバーの特性にある．

ファイルサーバーはファイルを保存するという基本的な機能を持つ．ファイルシステム上にまた，ファイルサーバーは，表 1 のファイルサーバー機能要件とファイルサーバーの実例にあるようにファイルサーバーでは利用者数などの規模により差異があるが，ファイル数，ファイルサイズ，更新頻度，ファイル更新数，更新者数が多い．

2. 侵入検知システム

ファイルサーバーの完全性を維持するための仕組みとして，ホスト型 IDS(HIDS) が適用できないかと考えた．

HIDS の基本的な動作はサーバー上の各ファイル状態 (パーミッション，タイムスタンプ，ファイルサイズ，ハッシュ値など) を取得，記録する．HIDS を定期的 (cron による定刻起動) に実行し前回実行時の状態と比較することで，侵入 (削除，改ざん) を検知する．

OSS の HIDS として利用されている，AIDE でのハッシュアルゴリズム毎のスキャン時間の検証を行う[*1]．

実行結果は，表 2 となる．検証環境 (32 万ファイル) では 1 スキャンに 2〜5 時間かかる．比較として，同一検証環境の一部 (1 万ファイル，小サイズファイル) を置いたフォルダに対して AIDE を実行した場合，1 スキャンにかかる時間は 3 秒程度となり，HIDS のスキャン時間は使用するハッシュアルゴリズム，スキャン対象のファイル数とファイルサイズに比例して実行時間がかかる．

実環境ではファイルサーバーの規模によるが，検証環境での 1 スキャン実行時間の数倍以上になることが予測される．

1 スキャンあたりのスキャン時間長時間になる事により，以下の問題も発生する．

- スキャン中にスキャン完了した領域への削除，改ざんによるスキャン漏れ．
- HIDS 多重起動による，サーバー性能劣化．
- HIDS 多重起動抑止による，スキャン未実施時間の発生．

スキャン時間が長時間化することで，スキャン中の削除，改ざんを検知できないとその間のファイルサーバーの完全性，が担保できなくなる．このため，一般的な HIDS をファイルサーバーに適用した場合には削除，改ざんの検知はできるが，検知時間が長時間になる事により，実用的な検知では無いと考える．

一般的な HIDS ではサーバーのコンテンツファイル，システムファイル，ログの削除，改ざんを検知する事を主目的としているため，削除，改ざんされたファイルの復旧については考慮されていない．サーバーの完全性を維持するためには復旧も必要となる，ファイルサーバーでは一般的な HIDS ではファイルサーバーの完全性の維持には向かないと考える．

ファイルサーバーの完全性を維持するために，改ざん，削除を検知することにあわせて，ファイルを復旧する方法を検討と検証する．

3. 削除，改ざん検知の仕組みと，復旧について検討

3.1 ハッシュアルゴリズムの検証

ファイルの改ざんを検知する手法としては，暗号学的ハッシュアルゴリズムを使用し，ファイルのハッシュ値を記録し，ファイルのハッシュ値を比較することで改ざんの有無を検知する事ができる．ハッシュ値は使用するハッシュアルゴリズムと対象ファイルのサイズにより生成にかかる時間が変化する．表 3 のとおりに，ハッシュアルゴリズムとファイルサイズで時間が変化する[*2]．

ファイルの改ざん検知を行うのであれば，ファイル単位，フォルダ単位であれば，ハッシュ値を生成して保存し，必要に応じて確認する方法がある．しかし，ファイルサーバー全域をシステマチックにカバーする場合，数十万以上のファイルに対してハッシュ生成処理を行うため，ハッシュアルゴリズム，ファイルサイズ，ファイル数により数時間に及ぶと考えられ，実際に検証環境に対して，ハッシュアルゴリズムを変更し実施した場合のハッシュ生成時間は表 4 の通り，最も短時間で終了した SHA1 でも約 3 時間，SHA256 では 5 時間に生成にかかった．

ファイルサーバーに対してハッシュ生成を行った場合，ハッシュ生成に時間かかるため，削除，改ざんを検知するまでの間隔が空いてしまい，その間情報の完全性が維持できていないこととなる．

ファイルサーバーの完全性を維持するためには，対象領域の変更、削除を効率的に検知し，ハッシュ値生成も効率的に行う必要がある．

ハッシュアルゴリズムは，ファイルサーバー上のファイル改ざん検知のためだけに使用するが，安全性を維持するために SHA512 を利用する．

3.2 パーミッションの変更検知

ファイルの改ざんを検知するアルゴリズムとしてはハッシュ値を生成することで改ざんを検知できる．しかし，ハッ

[*1] 検証環境は A.1 に記載の通り

[*2] 100MiB 以下は 1 秒以下のため省略した．

表 1 ファイルサーバーの機能要件と実例

機能要件	ファイルサーバーの実例
ストレージサイズ	24TiB〜
ファイル数	数百万〜
ファイルサイズ	数 KiB〜数百 GiB
フォルダ階層	深い (20 階層以上)
ストレージ利用サイズ	12TiB〜
ファイルの更新頻度	頻繁かつ複数ファイル同時更新
ファイルの更新タイミング	不定，ただし，夜間など更新頻度が低い時間がある
ファイルの更新数	多い
ファイル更新者	多い

表 2 AIDE の実行時間

ハッシュアルゴリズム	実行時間
MD5	164m31s
SHA1	166m43s
SHA256	289m20s
SHA512	221m32s

シュアルゴリズムではファイルのパーミッションの変化については影響しないため，ファイルのパーミッションの書き換えを検知する方法が必要である．

ファイルシステムを直接操作し，パーミッション，ファイルに関する時刻 (書き込み，パーミッション変更)，inode 番号等を取得することも可能だが，今回 Unix/Linux 系 OS で標準実装されている stat コマンドを使用することで対応する．

検証環境のデータ領域全域に対しての stat コマンド実行時間は 40 秒程度のため，stat コマンドをそのまま使用しても問題は無いと考える．

3.3 ファイルシステムでの差分検知

サーバーのファイルシステム上の変更を検知する方法は少ない．ファイルサーバーの OS とミドルウェアで使用

表 3 ファイルサイズ毎の SHA でのハッシュ値生成時間 (秒)

ファイルサイズ	SHA1	SHA256	SHA512
100MiB	0.68	1.46	0.97
500MiB	3.56	7.19	4.76
1GiB	7.20	14.36	9.55
2GiB	14.43	28.92	19.06
4GiB	29.25	59.58	38.28
8GiB	57.89	118.17	76.32
16GiB	121.69	236.23	157.84
32GiB	275.41	471.95	313.45
64GiB	553.43	942.27	633.34

表 4 SHA でのハッシュ値生成時間

ハッシュアルゴリズム	実行時間
SHA1	189m24s
SHA256	316m37s
SHA512	224m15s

できる，変更を検知できる機能を調査した．ファイルサーバーの変更検知できる機能は以下が存在する．

カーネル API(inotify)

- Linux カーネル 2.6 以降であればカーネル API に変更検知するための inotify が追加されている．
- inotfy ユーティリティを使用することでファイル，フォルダの変更検知が行える．
- Linux に限定されることと，監視フォルダ毎に設定が必要なため，ユーザーであれば自由にファイル，フォルダの作成を行えるファイルサーバーでの変更検知には向かない．

Audit 機能

- セキュリティ機能として，カーネルのシステムコールの呼び出しをトラップし，ログ出力を行う．
- FreeBSD，Linux などに実装されている．

Samba の VFS_Audit ログ

- Samba の機能，VFS_Audit 設定に該当したアクセスをログ出力する．
- Samba 経由でのファイルシステムへのアクセスしか記録ができない．

ファイルシステムの機能

- snapshot を取得することができるファイルシステムでは差分を取得できる．
- CIFS/NFS/ローカル操作，問わずに差分を検知できる．
- ファイルシステムの機能として snapshot を生成でき，ユーティリティとして差分生成ができるファイルシステムを採用することで，ファイル，フォルダの削除，改ざんを検知することが容易となると考える．

今回，ファイルシステムの機能を利用する方法が対応 OS が多いと考え，ファイルシステムの機能を利用する方法とする．ファイルシステム以外の方法は以下の理由により採用しない．

Audit と Samba の VFS_Audit ログについては，ログファイルをパースする処理が必要となり，書き込みが多い場合にサーバーへの負担となる．また，Samba の VFS_Audit ログは Samba 経由のアクセス以外ではログは生成されな

いため，変更差分を検知できる範囲が狭くなる．カーネル API は OS により実装状況が違うため，今回対象としない．

3.4 削除，改ざんからの復旧

ファイルの復旧手段としては，定期的にバックアップしておいて，必要時に復元するという運用が一般的である．

ファイルサーバーにおいてのバックアップ手法は以下が考えられる．

dump ファイルシステムを dump し，別メディアに保存する．

rsync 別サーバー，バックアップ領域にコピー，同期を行う．

snapshot ファイルシステムの snapshot 実行時のストレージを保存する．

dump と rsync は一般的に使用されているが，バックアップに時間がかかることや，バックアップ効率の点から，snapshot を使用したバックアップが以下の点で有効と考え，削除，改ざんからの復旧手段とする．

- snapshot は現在のファイルシステム，ボリュームマネージャーでは実装されている．
- snapshot は snapshot 実行時のファイルシステムの内容を保持する．
- 保持することのできる snapshot 数はファイルシステムにより変わる．ファイルシステムをアンマウントする必要は無いため，ユーザーが利用できない時間は発生しない．

3.5 ファイルサーバー環境と HIDS 実行環境との分離

生成した snapshot，ハッシュ値，パーミッション情報は，機密性，完全性の観点から一般ユーザーから隔離する必要がある．

通常，システム領域や，特権ユーザー限定のフォルダに格納するが，ファイルサーバーの管理業務においては，HIDS の実行結果と snapshot データにアクセスできる必要は無い．容易にアクセスできる状態では，何らかのミスや，ファイルサーバーの管理権限を分割，委譲したユーザーの不正に対して脆弱となり，ファイルサーバーと HIDS の機密性，完全性が維持できないと考える．

実行環境分離については，ファイルサーバーとしてのオーバーヘッド，管理性を考慮し，コンテナ技術を利用する．コンテナ上で，ファイルサーバー機能を展開しファイルサーバーとする．コンテナを実行するホスト上で，snapshot 取得，ハッシュ値生成等を行い，一般ユーザーから隔離することで HIDS 実行結果，snapshot を保護する．

3.6 削除，改ざんの検知，復旧の方針

ファイルサーバーの削除，改ざんの検知，復旧の機能は以下を採用する．

ストレージの変更検知
ファイルシステムの snapshot 機能

パーミッションの変更検知
stat コマンド

改ざん検知
SHA512

削除，改ざんからの復旧
ファイルシステムの snapshot 機能

snapshot，差分情報，ハッシュ値保護
コンテナ

4. ファイルシステムと OS

4.1 ファイルシステム

表 5 に Unix/Linux 系 OS で使用されるファイルシステムを調査した．

非商用 OS で容易に入手できる OS に実装されたファイルシステムで，ファイルシステムレベルで使用中のシステムに対して影響 (ファイルシステムへのアクセス停止など) がなく，snapshot を大量に作成でき，snapshot 間の差分を取得できるファイルシステムは Btrfs，ZFS に絞られる．

Btrfs と ZFS は機能的には似ており，共に OSS の OS で利用されているが，ファイルシステムとしての利用実績，筆者が使用している OS の標準機能として採用されている，ZFS[*3] をファイルシステムとして採用する．

4.2 コンテナ

コンテナについては，Unix/Linux で各種実装がある．コンテナについての要求は以下となる．

- 複数のプロセスを実行できること．
- リソース分割が行えること．
- OS 標準であること．
- 管理しやすいこと．

コンテナをサポートしている OS とコンテナ機能は表 6 の通り．

4.3 OS

ZFS をファイルシステムとしてサポートしている OS は表 7 の通り．

本研究のサーバー OS として，以下の理由により FreeBSD を採用する．

- 標準環境で ZFS が利用できる．
- ZFS の実績が FreeBSD7-RELEASE(2008 年) からと Solaris に次いで長い．
- jail(コンテナ機能) が利用できる．
- Pkg/Ports といった機能により，3rdparty 製ソフトウェアの導入が容易．

[*3] 2018 年現在 Solaris 以外の実装は OpenSolaris の ZFS からフォークした OpenZFS となっている．

表 5　ファイルシステム

ファイルシステム	対応 OS	スナップショット生成	差分確認機能	保持数制限
UFS	Solaris	可能	無し (snapshot をマウントし diff)	あり
UFS2	FreeBSD	可能	無し (snapshot をマウントし diff)	あり
EXT2,3,4	Linux 系 OS	LVM との併用で可能	無し (snapshot をマウントし diff)	あり
XFS	Linux 系 OS	LVM との併用で可能	無し (snapshot をマウントし diff)	あり
Btrfs	Linux 系 OS	可能	有り	なし
ZFS	Solaris,FreeBSD,Linux 系 OS	可能	有り	なし

- OS がメジャーバージョンアップしても使い勝手の変化が少ないため，長期間の運用が行いやすい．

5. 検証

サーバーを用意し，ファイルサーバとして運用を行えるように準備する．

5.1 検証環境

OS　OS を OS 用 HDD にインストールする．インストール後，OS，アプリケーションは最新のパッチを当てる．

アプリケーション　Ports から表 A·1 記載のアプリケーションをインストールする．

5.2 データ領域の準備

検証用データを格納するデータ領域のストレージを ZFS で用意する．

ZFS ストレージプールを二本の HDD でミラーモードで作成をする．

```
#zpool create tank mirror ada1 adb1
```

ZFS ファイルシステム作成する．

```
#zfs create tank/export
```

今回 atime を使用しないため，atime を無効化する．

```
# zfs atime=off tank/export
```

表 6　コンテナ対応 OS

OS	コンテナ機能名称
Solaris11	SolarisZone
CentOS	Docker
	LXC,LXD
Ubuntu	Dcoker
	LXC,LXD
FreeBSD	Jail

表 7　ZFS 対応 OS

OS	対応状況	備考
Solaris11	標準	商用，HW の対応がシビア
CentOS	サードパーティ	ZFS on Linux
Ubuntu	標準	
FreeBSD	標準	

atime が無効化されたことを確認する．

```
# zfs get atime tank/export
```

ここまでで，ファイルサーバー上に ZFS のストレージ領域が用意された．このストレージ領域に検証用データをコピーする．

5.3 snapshot

ZFS 上に保存されたデータをスナップショットを取得する．ZFS で snapshot を作成する．

```
#zfs snapshot tank/export@YYYYMMDDhhmmssN
```

snapshot は 1 秒以下で作成される．

```
0.000u 0.003s 0:00.69 0.0%␣␣0+0k 12+0io 0
    pf+0w
```

5.4 snapshot 間の差分生成

ZFS snapshot 間の差分を確認する．

```
#zfs diff tank/export@YYYYMMDDhhmmssN-1
    tank/export@YYYYMMDDhhmmssN
```

実行時間は 1 秒程度

```
0.000u 0.050s 0:00.82 6.0%␣␣115+168k
    655+0io 0pf+0w
```

zfs diff コマンドを実行することで，ファイルの削除，変更が有ったことが分かる．

```
#zfs diff tank/export@YYYYMMDDhhmmssN-1
    tank/export@YYYYMMDDhhmmssN


M   /export/TEST_DATA
+   /export/TEST_DATA/dummy_64GiB2
```

ZFS diff コマンドによる snapshot 間の変更状態は以下の通り．

zfs diff コマンドの実行結果に” M”,” -”,” R”，同じファイルに対して”-,+”が有った場合，改ざん，削除の可能性があると判断する．

表 8 ZFS snapshot diff 結果

ファイル/フォルダの変更	記号
前回から変更	M
前回あり，今回無し	-
前回無し，今回あり	+
名前変更	R
パーミッション変更	M
ファイルを削除，再作成	-,+

5.5 snapshot 間の差異が有る場合

前章の zfs diff コマンドで snapshot 間で差異がある場合，その差異にあわせて，処理を行う．

表 9 変更に対して検知後処理

変更記号	処理
M	ハッシュ，パーミッション確認
-	復元確認
+	処理無し
R	ハッシュ，パーミッション確認
-,+	ハッシュ，パーミッション確認

5.5.1 ハッシュ値生成

ZFS の snapshot は，以下に zfs snapshot は差分情報を保存している．

```
/export/.zfs/snapshot/YYYYMMDDhhmmssN
```

ZFS の snapshot diff で比較し，ファイル，フォルダに変更があった事を検知した場合，このフォルダのファイルに対して，ハッシュ値生成処理を行い事で，ファイル改ざんを検知することができる．

ファイルに対してハッシュ値生成を行う場合，数 GiB 以上のファイルの場合ハッシュ値生成にかかる時間が長くなる．

5.5.2 パーミッション情報

zfs の snapshot は，以下に snapshot 実行時の情報を保存している．

```
/export/.zfs/snapshot/YYYYMMDDhhmmssN
```

このフォルダのフォルダ，ファイルに対して，stat コマンドの実行を行う事で，ファイルパーミッションの改ざんを検知することができる．

stat の実行時間はファイルサイズによる実行時間の差異は無く，おおむね 1 秒以下で生成される．

5.6 コンフィグレーション

zfs の snapshot は 1 秒以下に行えるが，ファイルシステムと差分確認の負荷を減らすために 5 分程度にする．

また，ハッシュ値はファイルサイズに影響されて生成時間が変わるため，差分があったファイルをマークし，ハッシュ値生成を実行する．

6. 考察

大容量ストレージを持つファイルサーバーに HIDS を適用した場合，実用的な時間内にスキャンを行う事ができなかった．

一般的な HIDS は Web サーバー，サーバーの OS 領域への侵入を検知 (改ざん検知) するためのシステムであり，大容量ストレージ，サイズの大きなファイルのスキャンを対象としていないため，ストレージ領域を順次スキャンし，時間がかかっても問題が無かったと考える．

ファイルサーバーは，一般的な Web サーバー等とは違い，大容量ストレージを持ち，大量で，大小様々なファイルをスキャンするため，順次スキャンではストレージ上にある，ファイルすべてに対してスキャン処理を行うため，変更が無い多数のファイルに対してもスキャン処理が行われるため，効率が悪く，スキャンの長時間化に繋がっている．

スキャン時間の短縮のためには，大容量ストレージから変更のあったファイルだけを検知し，ハッシュ値生成の対象を絞る必要がある．

ハッシュ値生成をファイルサーバーのファイルシステムの snapshot と差分検知機能で検知した変更のあったファイルに対して実行することで短縮できると考えた．

この方法を検証環境に対して適用した結果，一般的な HIDS の実行結果は表 2 に記載の通り，2 時間〜5 時間かかっていたが，ファイルシステムの機能 (ZFS の snapshot と diff) を使用した場合，表 10 の時間で差分を検知する事が可能となり，HIDS で削除，侵入検知を行う場合の約 1/20 の時間での差分を検知できた．

HIDS でファイルサーバーの削除，侵入検知を行う場合は，ファイルシステム上の変化を効率よく検知する事が重要と分かった．

表 10 ファイルシステムの機能活用した場合の時間

操作	実行時間	備考
差分検知	1s	
パーミッション走査	1s	
ハッシュ値生成	11m	ファイルサイズと，ハッシュアルゴリズムに影響される．64GiB のファイルに対しての時間 SHA512

また，一般的な HIDS では削除，改ざんの検知が目的となっており，復元には，HIDS とは別に行われたバックアップからの復旧となる．このため，バックアップ後に作成されたファイルは復元できない．常にファイルの追加，変更が行われているファイルサーバーでは日に一度のバックアップでは，情報の喪失となり，完全性の維持ができなく

なるが，ファイルシステム機能 (ZFS の snapshot) を利用した HIDS では，ZFS の snapshot 保持数制限が無いことを利用し，数分単位で snapshot を生成しすることで，検知した時点の前回実施した snapshot から復元できるため，ファイルの復元の可能性を高くできると考える．

ファイルサーバーの完全性維持は難易度が高かったが，最新のファイルシステムの機能を利用することでファイルの削除，改ざんを実用的な時間で検知できるようになり，また，HIDS ではできなかった，削除，改ざんからの復元もできるようになり，ファイルサーバーの完全性の維持に貢献できると考える．

7. 追加研究事項

HIDS をファイルサーバーに適用して実用的な時間での検知や復元が可能かを検証をするには，すでに行ってきた調査の他に，追加の調査も必要となる．

7.1 異常状態の検知と対応

今回の研究では，ファイルサーバーにおいて削除，改ざんを検知と復旧することに注目して検討した．実際の運用においては，正常な変化なのか，異常な変化なのかを判定することが必要となる．正常，異常な変化かを判断する方法を検討したい．

7.2 大容量環境での実験

今回は，2TiB のストレージを用意し，データとして約 1TiB のデータを用意したが，企業で利用される環境に比べてストレージサイズ，ファイル数が少ないため，ファイル数，ファイルサイズを増やして検知にどのような影響があるか検証したい．

7.3 暗号学的ハッシュアルゴリズムについて

ハッシュアルゴリズムのスループットでハッシュ値生成にかかる時間が変わるため，以下の検証を行いたい．

- ハッシュアルゴリズムを GPGPU(CUDA/OpenCL), FPGA に対応させたハッシュアルゴリズムの作成と実行時間の検証
- ハッシュアルゴリズムとして，BLAKE2 等新しいアルゴリズムを適用した場合のハッシュ値生成の比較

7.4 snapshot の生成頻度，間隔，長期間の運用について

今回の検証では，数ファイルの操作と数回の snapshoto 生成で検証を行ったが，実環境で運用する場合，snapshot の生成頻度，間隔，保持期間について指針が必要と考える．長期間運用した際に snapshot がどのようにストレージを消費するかも調査し，snapshot 保持期間の指針を作成したい．

7.5 検証ハードウェアの更改

今回，手持ちの家庭用 PC サーバーで検証したため，さらに企業等で利用する環境に近づけるために以下の点でハードウェアの更改を行いたい．

CPU
　コア数，クロックの影響．

メモリ
　メモリ容量の影響．

ストレージキャッシュ
　ZFS のキャッシュ (ARC/L2ARC/ZIL) の影響．

RAID
　ソフト RAID のモード (1,5,6) 違いの影響

大容量ストレージ
　さらなる大容量，多量なファイルの影響．

付　　録

A.1 検証環境

A.1.1 OS

`FreeBSD-11.1-RELEASE-p10`

本環境では，ZFS のキャッシュ追加をはじめとした，OS のチューニングは行っていない．

A.1.2 ソフトウェア

OS インストール後に Pkg と Ports を最新に更新を行い，アプリケーションをインストールする．

表 A·1 ソフトウェア

	ソフトウェア名	バージョン
HIDS	AIDE	0.16
スクリプト言語	Python3	3.6
コンテナ	qjail	5.4
DB	SQLite3	3.23.1
Window ファイル共有	Samba4	4.6.15

A.1.3 AIDE コンフィグレーション

AIDE のコンフィグファイルは以下のように設定し，ハッシュ関数の評価毎に変更し実行した．

```
database=file:///var/db/aide/databases/
    aide.db
database\_out=file:///var/db/aide/
    databases/aide.db.new
/export          p+i+n+u+g+s+m+sha1
#/export        p+i+n+u+g+s+m+sha256
#/export        p+i+n+u+g+s+m+sha512
#/export        p+i+n+u+g+s+m+md5
```

A.1.4 検証データ

表 A·2 検証データ

ストレージサイズ	2TiB
RAID	softRAID(ZFS Mirror-Mode)
データサイズ	1TiB
ファイル数	313,237
データ構成	FreeBSD,OpenBSD,NetBSD の ISO イメージ, Pkg データ等. 1KiB〜64GiB サイズのランダムデータファイル

ランダムデータは以下のようにして生成した.

```
#dd if=/dev/urandom of=/export/TEST_DATA/
     bs=1k count=1
```

A.1.5 ハードウェア

表 A·3 検証用ハードウェア

	スペック
ベース	HPE ML110G7
CPU	Intel Corei3 2100 (2C/4T 3.1GHz)
メモリ	16GiB (EDDR3-1333 CL9 ECC)
HDD(OS)	250GiB (SATA 6Gb/s 7200rpm)
HDD(DATA)	2TiB*2 (SATA 6Gb/s 7200rpm)
グラフィックカード	Nvidia GeForce GTX750Ti(2GiB)

参考文献

[1] FreeBSD オフィシャル
入手先 ⟨http://www.freebsd.org⟩ (参照 2018-04-05).
[2] Oracle Solaris ZFS 管理ガイド
入手先 ⟨https://docs.oracle.com/cd/E24845_01/html/819-6260/gbciq.html⟩
ZFS(Wikipedia)
入手先 ⟨https://ja.wikipedia.org/wiki/ZFS⟩ (参照 2018-04-05).
[3] AIDE
入手先 ⟨http://aide.sourceforge.net/⟩ (参照 2018-04-28).
[4] 情報セキュリティ
入手先 ⟨https://ja.wikipedia.org/wiki/情報セキュリティ⟩ (参照 2018-04-05).
[5] 暗号学的ハッシュ関数
入手先 ⟨https://ja.wikipedia.org/wiki/暗号学的ハッシュ関数⟩ (参照 2018-04-05).
[6] 電子政府推奨暗号リスト
入手先 ⟨https://www.cryptrec.go.jp/method.html⟩ (参照 2018-06-06).
[7] inotify
入手先 ⟨https://qiita.com/stc1988/items/464410382f8425681c20⟩
入手先 ⟨https://www.ibm.com/developerworks/jp/linux/library/l-inotify/⟩ (参照 2018-04-05).
[8] Btrfs
入手先 ⟨https://ja.wikipedia.org/wiki/Btrfs⟩ (参照 2018-05-25).
[9] Samba オフィシャル
入手先 ⟨https://www.samba.org⟩ (参照 2018-06-04).
[10] samba VFS Audit
入手先 ⟨https://www.samba.org/samba/docs/current/man-html/vfs_full_audit.8.html⟩ (参照 2018-06-04).
[11] backup
入手先 ⟨https://ja.wikipedia.org/wiki/バックアップ⟩ (参照 2018-06-04).