IIR

Internet
Infrastructure
Review

Jun.2022
Vol.55

定期観測レポート

今、求められるメールセキュリティ ～パスワード付きZIPと送信ドメイン認証「DMARC」

フォーカス・リサーチ(1)

mac_aptプラグインの作成(後編)

フォーカス・リサーチ(2)

サービス基盤でのストレージに関する課題と対応

IIJ
Internet Initiative Japan

Internet Infrastructure Review

June 2022 Vol.55

エグゼクティブサマリ

2022年3月に発行された本レポートのVol.54（https://www.iij.ad.jp/dev/report/iir/054.html）で、2014年にロシアがクリミアを併合した際のインターネットの接続性の変化について分析した「フォーカス・リサーチ」を掲載しました。2月24日から始まったロシアによるウクライナ侵攻の最中に当該記事を掲載することになりましたが、発行から約1カ月経過した5月1日、ロシアが占領した地域のインターネット接続が、数時間の通信断の後、ロシアの通信事業者経由で回復したとの観測結果がNETBLOCKS社から発表されました*1。

クリミアの件を「IIR」で掲載した直後というタイミングに驚くと共に、2014年とは異なり、ネットワークの変更が非常に早い時期に行われたことにも改めて驚きました。8年前よりもインターネットで流入する情報のコントロールの重要性が増している証左かもしれません。

戦争の世紀であった20世紀が終わった今、このような事態が起きていることに衝撃を受けています。安全保障の文脈における技術の利活用について見直しの必要性が叫ばれていますが、私たちが担っている情報通信技術においても同様です。あらゆる技術が私たちの生活を豊かにするために使われることを切に祈り、行動していきたいと考えています。

「IIR」は、IIJで研究・開発している幅広い技術を紹介しており、日々のサービス運用から得られる各種データをまとめた「定期観測レポート」と、特定テーマを掘り下げた「フォーカス・リサーチ」から構成されます。

1章の「定期観測レポート」は、メッセージングです。1年前のメッセージングの定期観測レポートにおいて、Emotetが自分自身をZIPで暗号化して猛威をふるっていることを紹介しました。日本ではメールにファイルを添付する際、ZIPで暗号化して、パスワードを別送する手法が広まっており、社内へのEmotetの侵入を防ぐことを困難にしています。IIJでは2022年1月、メールに添付された暗号化ZIPファイルを原則受信拒否することにしました。また2021年12月には、送信ドメイン認証技術であるDMARCのポリシーを強化しています。これらは、IIJ自身のメールシステムのセキュリティを強化する施策であり、その経緯や課題についてまとめた本レポートは、組織においてメールシステムを管理する皆様に大いに参考にしていただけると思います。

2章の「フォーカス・リサーチ」では、前号に引き続きmac OS用フォレンジック解析フレームワークとして開発されているmac_aptにおけるプラグインの作成について解説しています。今回はmac_aptで保存されるデータの内容を説明すると共に、実際にプラグインの設計と実装について述べています。前回のレポートと合わせてご一読いただけましたら幸いです。

3章の「フォーカス・リサーチ」は、IIJのサービス基盤におけるストレージに関する課題と対応について取り上げています。コンピュータの処理能力の増大、ネットワークのデータ転送速度の向上に支えられ、全世界で生成・処理されるデータ量は大きく伸長しています。それらのデータを蓄積しているのがストレージであり、ストレージもコンピュータやネットワークと同様に大きく進化しています。IIJでも増大するデータを安全に保管するために多くのストレージを運用しています。本レポートではストレージの基本的な説明に加え、IIJで多く利用しているFC-SANとストレージの運用技術について紹介します。

IIJは、このような活動を通してインターネットの安定性を維持しながら、日々、改善・発展させていく努力を行っています。今後も企業活動のインフラとして最大限にご活用いただけるよう、様々なサービスやソリューションを提供し続けてまいります。

島上 純一（しまがみ じゅんいち）

IIJ 常務取締役 CTO。インターネットに魅かれて、1996年9月にIIJ入社。IIJが主導したアジア域内ネットワークA-BoneやIIJのバックボーンネットワークの設計、構築に従事した後、IIJのネットワークサービスを統括。2015年よりCTOとしてネットワーク、クラウド、セキュリティなど技術全般を統括。2017年4月にテレコムサービス協会MVNO委員会の委員長に就任、2021年6月より同協会の副会長に就任。

＊1　NETBLOCKS（https://netblocks.org/reports/internet-disruptions-registered-as-russia-moves-in-on-ukraine-W80p4k8K）。

今、求められるメールセキュリティ ～パスワード付きZIPと送信ドメイン認証「DMARC」

1.1 はじめに

昨年6月に発行した本レポートのVol.51（https://www.iij.ad.jp/dev/report/iir/051/01.html）の「定期観測レポート」で、迷惑メールと、ウイルス数の集計と傾向を報告しました。特徴的だったのは、最大値でその前年度の200倍もの迷惑メールを受信したことと、Emotet（エモテット）と呼ばれるウイルスが自分自身をZIPで暗号化することでウイルススキャンを回避し、猛威をふるっていたことでした。

今回は、IIJがそうした脅威から自社を守るために取り組んだ2つのセキュリティ強化策について紹介します。1つは暗号化ZIPの運用を廃止したこと、もう1つはDMARCの強化です。読者の皆様にもぜひ実施いただきたい内容ですので、ご参考になれば幸いです。

1.2 IIJ、暗号化ZIPの運用を廃止

1.2.1 暗号化ZIP廃止の背景

IIJでは、2022年1月26日以降、メールに添付された暗号化ZIPファイルを原則受信拒否するよう、全社ポリシーを変更しました[*1]。

日本国内では、メールで添付ファイルを送付する際にパスワード付きZIPとして暗号化し、そのパスワードを別送する手法が誤送信対策として広まっています[*2]。しかし、この手法は誤送信対策として効果が少ないだけでなく、ウイルススキャンを回避できてしまう致命的な問題を抱えていることから、CISA（米国サイバーセキュリティ・インフラセキュリティ庁）からも受信拒否することが推奨されています[*3]。

また、前途のとおり、猛威をふるったEmotetがこの手法を用いており、今後も同様の手口で拡散するウイルスの登場が予想されます。IIJ社内のみならず、大切なお客様や取引先様よりお預かりする情報を守るためにも、このリスクを看過することはできないとの経営判断に至り、トップダウンで対策が進められました。

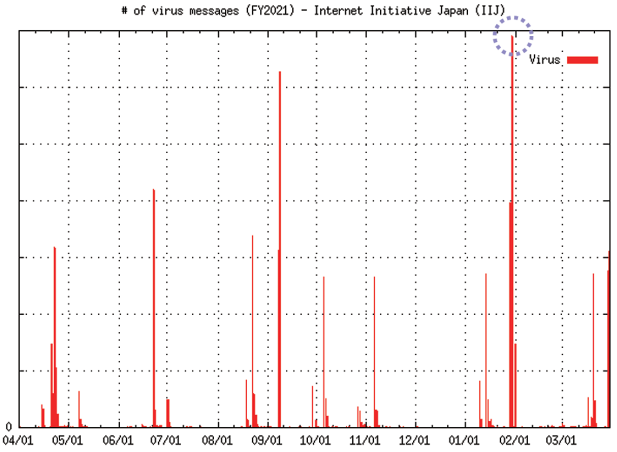

図-1 IIJハニーポットに着信したウイルス（2021年4月～2022年3月）

1.2.2 暗号化ZIP廃止までの準備

今回のポリシー変更はおおよそ次のような手順で進められました。

- 情報システム部門から危機管理部門・経営層への説明
- 経営層から社内へのリスク説明・方針説明
- 情報システム部門での対策実現のための検討
- 情報セキュリティ担当部門での統一ルール検討
- 社内の各部門への説明とスケジュールの展開
- お客様・取引先様への説明
- ポリシーの変更

経営層からの方針説明があってから最終的にポリシーを変更するまでに、約1年掛かりで準備が進められました。丁寧に準備を進めていった結果、約半年経過した現時点で大きな混乱は生じていません。

1.2.3 暗号化ZIP廃止の効果

偶然にも、暗号化ZIPを原則受信拒否とした日の週末、テイクダウンされたはずのウイルス「Emotet」が復活していることが確認されました。図-1のとおり、IIJが運用しているハニーポットでも1月末に顕著に表れています。IIJでは、ウイルスを受信することなくゲートウエイで破棄されていますので、早くも大きな

＊1　IIJからのプレスリリース「パスワード付きzipファイルが添付されたメール及び別送のパスワード記載メール（PPAP）に対する当社運用の変更について」（https://www.iij.ad.jp/ppap/）。
＊2　PPAPと呼ばれることがあります。
＊3　CISA「Emotet Malware」（https://www.cisa.gov/uscert/ncas/alerts/aa20-280a）。

効果を上げていることが確認できました。社内ネットワークは安全に守られたのです。

この一連の方針転換は、実施するまでに社内の様々な部門、例えば、危機管理部門、情報システム部門、営業部門、広報部門、といった、多様な利害関係者がいる中で調整が進められました。長らく取ってきた手法を転換することはときに痛みを伴うこともありますが、リスクを先送りすることでは解決できません。重大事が起こる前に、早急な対策をお勧めします。

1.2.4 代替策はあるか

暗号化ZIP廃止論とセットで話題になるのが、この代替策です。前回のIIR Vol.51では、どのような手法を用いても一長一短があることを説明しました。

IIJでも、この点を十分理解しており、社外とのファイル共有方法として、3つの手段を用意しています。

1つ目は、全社統一のオンラインストレージです。代替策として論じられている、最もオーソドックスな手段でしょう。ただし、オンラインストレージにはメールをアーカイブしているときに、後からファイルの追跡がしにくいという内部統制上の弱点になり得る点や、URL1つで大容量のファイルを持ち出すことができてしまうため内部犯行を見つけ出すのが困難である点をリスクとして抱えています。

2つ目は、添付ファイルをそのまま送信する方法です。暗号化ZIPは誤送信対策として効果が少なく、ウイルススキャンを回避できてしまうことが問題なのですから、そのまま送信すれば良いという判断です。また、一部の取引先様によっては、宛先の社内から社外Webへのアクセスが許されていないケースもありますので、そのような場合は直接送信して対応しています。

3つ目は、従来の暗号化ZIP運用を続ける方法です。これは例外措置です。メールでのやり取りは、送信・受信の双方あって成立するものですから、一部の組織や取引先様で従来の手法を取らざるを得ないケースが残っています。このような場合は、リスクを十分に理解した上で、その組織長の承認によって、例外措置を入れた運用をしています。

そして、この3つの方法をメール監査システムと組み合わせることで、それぞれの弱点を補強しています（表-1）。

繰り返しになりますが、暗号化ZIPを一律受信拒否としたことで、Emotetを代表とするウイルスの脅威にさらされることがなくなりました。こうしている間にも攻撃者は次のターゲットを狙っています。全社ポリシーを変更するには時間が掛かることもありますので、今すぐ対策に着手されることをお勧めします。

1.3 IIJ、DMARCポリシーを強化

1.3.1 DMARCポリシーを強化した背景

IIJは、2013年にDMARCポリシーを導入し、しばらくの間p=none、つまり、外部に対してDMARC検証が失敗したメールについて何もしないでください、と宣言してきました。p=noneだとしても、DMARCレコードを公開してDMARCレ

	アドバンテージ	リスク・問題点		
1. オンラインストレージを利用	・ストレージ側でウイルススキャンが可能 ・誤送信対策として有効なケースがある ・大容量のファイルを送受信可能	・証跡が追跡しづらいため内部統制上の弱点になる ・ファイルを持ち出す内部犯行の検出が困難	＋	メール監査システム
2. 添付ファイルをそのまま送信	・ゲートウェイ側でウイルススキャンが可能 ・追加設備・投資が不要 ・宛先がどのような環境でも適用できる	・誤送信対策がない		
3. 従来の暗号化ZIP運用を継続（例外措置）	・従来の方法を変えずに済む	・ウイルススキャンを回避するため無防備でリスクが大きい ・宛先によっては受信拒否される ・受信側の手間・負担が大きい ・業務効率化・自動化を妨げる		

表-1 代替策の代表的なアドバンテージとリスク

ポートを受信することが可能です。受信したレポートの統計データを集計することで、なりすましメールを見抜くことができるため、自社ドメインのブランドを守るという観点ではとても有用です。IIJが提供しているメールセキュリティSaaS「IIJセキュアMXサービス」でも、DMARCレポートを自動で集計し、統計情報を参照できる機能を提供しています。

近年、スパムフィルタを巧みに回避するようなフィッシングメールや差出人詐称メールが流行しており、メールによる様々な脅威に対して多角的なアプローチが必要になっています。法人系メールセキュリティSaaSをお客様に提供しているIIJとしても、まずは社内メールのセキュリティ強化を図るため、p=quarantineへポリシーを変更することとなりました。

1.3.2 送信ドメイン認証

IIJのDMARC導入について報告する前に、改めて送信ドメイン認証技術について以下に記します。送信ドメイン認証技術にはSPF、DKIM、DMARCが世界的に一般に使用されており、それぞれRFCにて定義されています（表-2）。

DMARCポリシーには、表-3の3つが指定できます。DMARCポリシーで重要なのは、送信者から受信者に対して"こうしてください"とお願いするだけであって必ず宣言したとおりに受信者側システムで取り扱われる訳ではない点です。受信側のメール解析システムにDMARCを評価するような機能、フィルタが実装されていなければ何の影響もありませんが、企業としてDMARCポリシーを外部に対して宣言することで自社ドメインの正当性を表明することができます。

いままでのメールに関するフィルタ処理は受信者側での判断に委ねられていましたが、DMARCポリシーを使ったフィルタ処理は、送信者側から"DMARCの認証に失敗したメールはこう処理してほしい"と受信者へ依頼をする形となっており、メールをフィルタする手段としては従来にはなかった画期的なフレームワークです。

1.3.3 送信ドメイン認証導入の準備

DMARCポリシーを変更するにあたって、作業自体はDMARCレコードの変更のみですが、それ以前にSPFレコードの宣言またはDKIM署名の実装が必要です。DMARCポリシーはSPFまたはDKIMを元にそのメールが正当かどうかを評価しますので、送信ドメイン認証されていないメールが社内ユーザから送信される状況を作らなければいいわけです。

そこで大事なのが、次のような下準備です。

① 会社のメールは会社のメールアドレス（ならびにメールシステム）からしか出してはいけない、という社員への啓蒙活動
② メールの出口の集約による送信ドメイン認証への対応コスト削減
③ （DMARCポリシー導入後の）定期的なDMARCレポートの確認

これら３つの対策について詳述します。

p=の値	dmarc=failした時に受信者に
none	何もしないようお願いする
quarantine	隔離するようお願いする
reject	拒否するようお願いする

表-3 DMARCポリシー

送信ドメイン認証技術	RFC	概要
SPF	7208（注1）	SPFレコードを宣言することで、記載のIPアドレスからの送信メールが正規メールであることを外部に宣言する
DKIM	6376（注2）	メールに電子署名することで送信メールの内容の改ざんがされているかを検証することができる
DMARC	7489（注3）	SPFならびにDKIMの検証が失敗したメールについて、受信者に対してどう取り扱ってほしいかを送信者側が表明する

注1：https://datatracker.ietf.org/doc/rfc7208
注2：https://datatracker.ietf.org/doc/rfc6376
注3：https://datatracker.ietf.org/doc/rfc7489

表-2 一般的な送信ドメイン認証技術の特徴

■ ①社員への啓蒙活動

IIJではiij.ad.jpドメインで送信するメールが何種類か存在します。

- IIJ社員が送信する業務メール
- 各システム機器からの通知メール
- お客様へ送信するアナウンスメール

社員の業務メールについては、以前は社内の様々なサーバから送信されている状況でしたが、社内メールシステムの出口が情報システム部門によって管理されるようになったことと、そもそも社内メールシステム以外から@iij.ad.jpを使ってメールを送信することをポリシーにより禁止し、社内からインターネットへの通信を常時監視して定期的に違反ユーザへ送信停止依頼を通知することで解消しました。

また、社内の様々なシステムから送信されるメールについても問題がありました。IIJではいくつものサービスが各部署で運用されており、至るところからアラートメールや通知メールなどが送信されていました。現在は、サービスを統括する部署により各種サービスから送出されるすべてのメールの出口が統一されています。

更に、グループ会社や地方拠点の中にもアラートメールなどの差出人アドレスをiij.ad.jpとしている機器があったので、各担当者へ通知をしてIIJで定めたポリシーに従ってもらいました。

■ ②メールの出口の集約

前述したように、"メールの出口の集約化"というのは送信ドメイン認証の運用を考慮する上で非常に重要になってきます。

SPFレコードにはメールの送信元IPアドレスをCIDR表記することができますが、メールの出口を/32や/31、広くなっても/28程度に取りまとめておけばレコードを長々と書く必要もなく、出口が追加や変更になったとしても運用コストを削減することができます。メールの出口となるIPアドレスがいくつもある場合、includeを重ねに重ね、SPFの検証がうまくできなくなったり、考慮漏れが発生して意図せずにSPF検証に失敗したりする場合もあります（ちなみに、RFC 7208にはSPFにincludeを記載する際、DNS lookupの回数は10回までとの記載があります）。

■ ③定期的なDMARCレポートの確認

IIJでは、各所からDMARCレポート（rua）を受信しています。様々な組織から"我々のシステムで受信した@iij.ad.jpからのメールの送信ドメイン認証の結果はこのとおりです"という情報が定期的に送られてくるのです。メール送信の出口を管理しているため、それ以外から送られているメールについては基本的にはスパムメールのはずですが、以下のようなそうではないメールも散見されたため、その情報を元に少しずつ各部署にポリシー設定の修正をお願いしていきました。

- アラートメール
  ネットワーク機器やサーバの監視システムなどのenvelope fromを勝手にiij.ad.jpにしてSPF/DMARC failしているケースがあった
- プロモーションならびにリクルーティングメール
  これらは外部のSaaSを利用することがよくあるが、そのシステム内でenvelope fromがiij.ad.jpとなり、送信ドメイン認証を考慮せずに送っているメールがあった

図-2 iij.ad.jp 2022/02のDMARCレポートサマリー
（左）検証結果の割合（右）検証失敗ドメインランキング上位20ドメイン

1.3.4 DMARCポリシー変更の決断

DMARCポリシーは段階的にquarantineからrejectへと強化していくことも可能です。IIJでは、DMARCポリシーp=quarantineを導入することにしました。隔離であれば、メールを受け取った相手方でDMARCポリシーによるフィルタリングをしていたとしても、メールを受け取れないわけではないためです。

■ 実際の作業

作業自体はとても簡単で、DMARCレコードの宣言をp=noneからp=quarantineに書き換えるだけです。本報告では詳細の記載は省略しますが、この段階で既にSPFレコードの宣言、ならびにDKIM署名の実装が完了していることが前提となっています。

事後のレコード確認を含めても、ものの10分もかからずに作業は完了します(図-3)。

ここでDMARCレコードについて紹介しておくと、adkim、aspfパラメータはDKIM、SPFそれぞれについての認証識別子(ドメイン)のアライアメントモードをr(relaxed mode)またはs(strict mode)と指定することができ、該当メールの組織ドメインとHeader Fromが一致しているかどうかの判定をする際のパラメータとなります。

例えば、組織ドメインがexample.com、Header Fromがsystem@alert.example.comであるメールのSPFを元に評価する場合、アライアメントモードrを指定していると認証結果がpassとなります。また、sを指定していた場合、完全にドメインが一致している必要があるためfailとなります。ruaはDMARCレポートの送信先の指定であり、前項目で紹介した集計結果はdmarc-rua@dmarc.iij.ad.jpにて受信したレポートを可視化した情報となります。

■ 変更後の影響

IIJでは2021年12月15日にDMARCレコードを変更しました。

設定変更後、社内からの問い合わせ状況や動向を注視していましたが、大きな混乱はありませんでした。DMARCレコードの変更を難しいと考える方も多いかと思いますが、実際はそこまで影響もなくメールを受信した相手に対して"我々は送信管理をしたメールしか出していないのでそれ以外は隔離なり拒否してください"と宣言できるので、導入を検討してみてはいかがでしょうか。

過去には、IIJのハニーポット環境でDMARCポリシーを宣言していないことから大量のなりすましメールが送信されてしまう事象が観測されています。メール文書に正当性を保持したい機関(金融機関や行政機関など)では、DMARCを導入することで詐称メールと明確な差別化ができるのでお勧めです。

本稿を執筆中、IIJではDMARCポリシーp=quarantineを導入して数週間、様子を見ていましたが、特に業務メールに影響がないと判断し、2022年3月23日にDMARCポリシーをp=rejectに変更しました。DMARCポリシーp=quarantineを導入した際と同様、社内からの問い合わせなどもありませんでした。

```
$ dig _dmarc.iij.ad.jp txt

;; ANSWER SECTION:
_dmarc.iij.ad.jp. 3600 IN TXT "v=DMARC1; p=none; adkim=s; aspf=s; rua=mailto:dmarc-rua@dmarc.iij.ad.jp"

作業前のiij.ad.jpのDMARCレコード

$ dig _dmarc.iij.ad.jp txt

;; ANSWER SECTION:
_dmarc.iij.ad.jp. 3600 IN TXT "v=DMARC1; p=quarantine; adkim=s; aspf=s; rua=mailto:dmarc-rua@dmarc.iij.ad.jp"

作業後のiij.ad.jpのDMARCレコード
```

図-3 作業前と作業後のiij.ad.jpのDMARCレコード

1.4 送信ドメイン認証統計

1.4.1 送信ドメイン認証の普及状況

テレワークが世の中に広まりはじめて早2年が経ちました。2021年度はEmotetと呼ばれるウイルスを用いたメールによるサイバー攻撃が多く観測された1年でした。

2021年4月～2022年3月の期間、IIJが提供するメールサービスにて集計した送信ドメイン認証の結果とその割合を図-4～図-6に示します。

前回、本レポートのVol.51（https://www.iij.ad.jp/dev/report/iir/051/01.html）で報告した集計結果と比較すると、DKIMとDMARCのpass割合が増えていることが分かります。DKIMのpass割合が前回と比較して数ポイント増加していることから、リモートワークの状況下にて各企業でのSaaS導入がやや増加傾向にあるのではないかという予測ができます。また、DMARCのpass割合も増加していることから、徐々にではありますが、送信ドメイン認証への世間への関心が高まっている様子が窺えました。

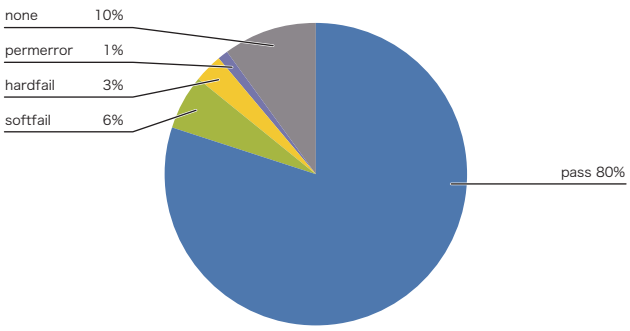

図-4 SPFによる認証結果割合

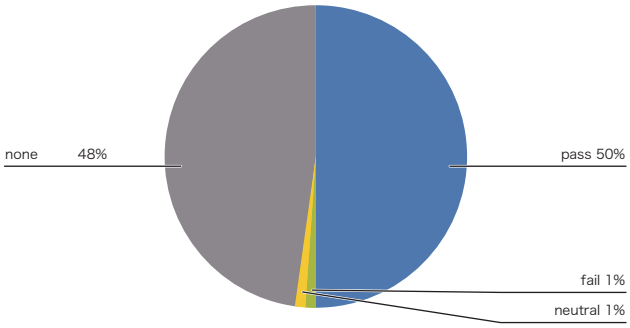

図-5 DKIMによる認証結果割合

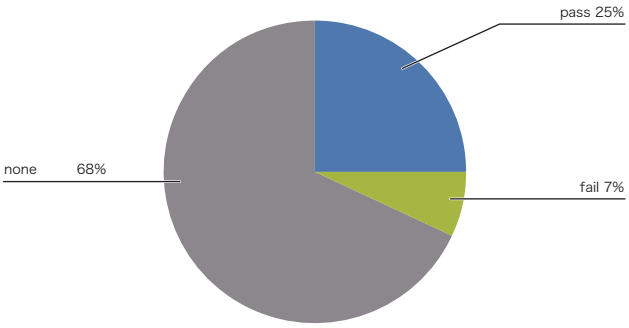

図-6 DMARCによる認証結果割合

執筆者：
古賀 勇（こが いさむ）
IIJ ネットワーク本部 アプリケーションサービス部 運用技術課 課長。
2007年IIJ入社。メールサービスの運用業務に従事し、現場でメールに関する動向を調査。お客様のメールボックスを守るため、最新の攻撃手法や、迷惑メールのトレンド、対策情報などを発信中。

執筆者：
今村 侑輔（いまむら ゆうすけ）
IIJ ネットワーク本部 アプリケーションサービス部 運用技術課 エンジニア。
2015年IIJ入社。メールサービスの運用業務に従事。IIJ Europeでの就業経験を活かし、日々グローバルに活躍中。

mac_aptプラグインの作成(後編)

IIR Vol.54ではmacOS用フォレンジック解析フレームワークmac_aptのデモプラグインでプラグインの基本的な構造を解説しました[*1]。今回は、前回の記事で最後に触れた「~/Library/Caches/<Application Bundle ID>/Cache.db」に保存されているデータの内容と、このアーティファクトを解析するmac_aptプラグインの実装について解説します。IIR Vol.54の記事を読んでいない方は、先にそちらを読んだ方が理解しやすいのではないかと思います。

2.1 Cache.dbに残る情報

まず、Cache.dbに記録されている情報を確認します。Cache.dbにはNSURLRequstなどのAPIを使用したHTTP/HTTPS通信のデータがキャッシュとして保存されています。Cache.dbファイルはSQLite形式のデータベースで、表-1～表-5に示すテーブルにデータが保存されています。キャッシュデータは基本的に、このデータベースに保存されますが、一定以上のサイズのデータはfsCacheDataディレクトリ以下に

カラム	データの種類
entry_ID	エントリーID
response_object	plist形式のBLOB。アクセス先URLやHTTPステータス、レスポンスヘッダなどが記録されている。
request_object	plist形式のBLOB。アクセス先URLやアクセスメソッド、リクエストヘッダなどが記録されている。
proto_props	不明(キャッシュの制御に使われている?)
user_info	不明(筆者が確認した範囲ではすべてNULL)

表-1 cfurl_cache_blob_data

カラム	データの種類
entry_ID	エントリーID
isDataOnFS	サーバからのレスポンスデータの保存形式を表すフラグ 0の場合、receiver_dataにサーバからのレスポンスボディが保存されている。 1の場合、ファイルとしてサーバからのレスポンスボディが保存されている。
receiver_data	サーバからのレスポンスデータ(isDataOnFS = 0)、または、それが保存されたファイル名(IsDataOnFS = 1)が記録される。 ファイルは「~/Library/Caches/<App Bundle ID>/fsCacheData/」以下に保存される。ファイル名はUUIDフォーマット。

表-2 cfurl_cache_receiver_data

カラム	データの種類
entry_ID	エントリーID
version	不明(筆者が確認した範囲ではすべて0)
hash_value	不明
storage_policy	不明(筆者が確認した範囲ではすべて0)
request_key	アクセス先のURL
time_stamp	URLにアクセスしたタイムスタンプ
partition	不明(筆者が確認した範囲ではすべてNULL)

表-3 cfurl_cache_response

カラム	データの種類
schema_version	スキーマのバージョン(筆者が確認した範囲ではすべて202、もしくはこのテーブル自体が存在しない)

表-4 cfurl_cache_schema_version

カラム	データの種類
cfurl_cache_response	cfurl_cache_responseのentry_IDの最大値?

表-5 sqlite_sequence

＊1　Internet Infrastructure Review(IIR)Vol.54 フォーカス・リサーチ(1) mac_aptプラグインの作成(前編)(https://www.iij.ad.jp/dev/report/iir/054/02.html)。

ファイルとして保存されます(図-1)。テーブル名が「cfurl_cache」で始まっていることから、「CFURL Cache」と呼ばれているようです。

この情報を確認することで、プログラムが通信を行った日時や通信先のURLだけではなく、サーバからのレスポンスも確認することができます。フォレンジック解析ではユーザやプログラムのアクティビティの履歴が非常に重要になります。したがって、このアーティファクトは非常に有益な情報源となります。

2.2 作成するプラグインの設計

2.2.1 取得対象データ

プラグイン作成に入る前に、アーティファクトからどのような情報を取得することができるか、もう少し詳細に確認しましょう。

表-1～表-3を見ると、通信を行ったタイムスタンプ、アクセス先のURL、クライアントからのリクエストメソッド及びヘッダ、サーバのHTTPステータス及びレスポンスヘッダ、サーバからのレスポンスボディを取得することができます。これらのテーブルはentry_IDをキーにして各データを紐づけることができます。更に、Cache.dbが保存されているファイルパスの中にある「<Application Bundle ID>」部分は通信を行ったプログラムを指しています。プラグインとしては、これらの情報を取りまとめて解析結果として保存すれば良いでしょう。

ところで、cfurl_cache_blob_dataテーブル(表-1)のresponse_objectとrequest_objectはplist形式のBLOB(図-2)が保存されています[*2]。このデータはplist形式のまま解析結果として保存するのでなく、解析者が内容を把握しやすいようにパースした結果を保存すべきです。

```
% ls -alR ~/Library/Caches/com.apple.osascript
total 128
drwxr-xr-x    4 macforensics  staff    128  2 10 17:36 .
drwx------+ 150 macforensics  staff   4800  4 27 15:12 ..
-rw-r--r--@   1 macforensics  staff  65536  5 19  2021 Cache.db
drwxr-xr-x@   4 macforensics  staff    128  2  2  2021 fsCachedData

/Users/macforensics/Library/Caches/com.apple.osascript/fsCachedData:
total 344
drwxr-xr-x@ 4 macforensics  staff     128  2  2  2021 .
drwxr-xr-x  4 macforensics  staff     128  2 10 17:36 ..
-rw-r--r--@ 1 macforensics  staff  116503 11  9  2020 2B1680C0-DAE0-4EA0-9EC0-C4FC7F86A8C0
-rw-r--r--@ 1 macforensics  staff   53755  2  2  2021 A391D5EC-9FCF-4993-A0AF-EEF2C871EF6A
```

図-1 ファイルの構成

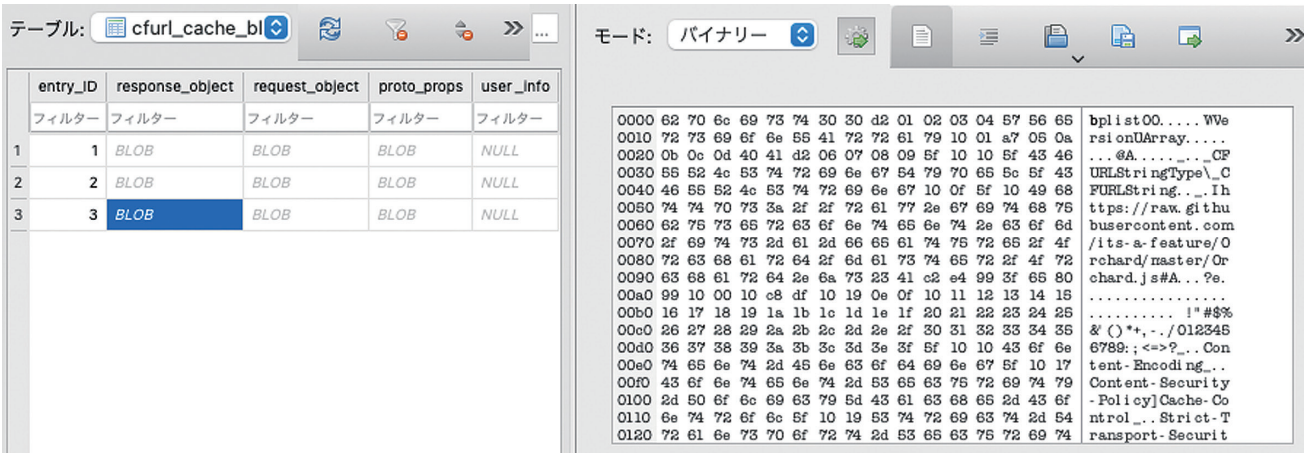

図-2 response_objectのサンプル

*2　先頭の「bplist00」はバイナリフォーマットのplistのマジックナンバーです。

DB Browser for SQLite[*3]などでrequest_objectのデータをエクスポートして、macOS標準コマンドのplutilで確認すると図-3のようになります。この中でフォレンジック解析に必要そうなデータは、「Array」の18番目と19番目のデータです。18番目はHTTPリクエストメソッドで、19番目はHTTPリクエストヘッダです。

19番目には「__hhaa__」にBase64エンコードされたデータも保存されています。このデータはバイナリフォーマットのplistであり、内容を確認すると図-4のようになります。内容的には、HTTPリクエストヘッダと同じものですので、これは解析結果に含める必要はありません。response_objectのデータについても同様に確認することができます(図-5)。こちらの場合、「Array」の3番目がHTTPステータスで、4番目がHTTPレスポンスヘッダです。

```
% plutil -p cfurl_cache_blob_data__request_object_2.bin
{
  "Array" => [
    0 => 0
    1 => {
      "_CFURLString" => "https://www.example.com/"
      "_CFURLStringType" => 15
    }
    2 => 60
    3 => 1
    4 => "__CFURLRequestNullTokenString__"
    5 => 1
    6 => 134
    7 => "__CFURLRequestNullTokenString__"
    8 => "__CFURLRequestNullTokenString__"
    9 => 1
    10 => 0
    11 => 0
    12 => 0
    13 => 0
    14 => 0
    15 => -1
    16 => "__CFURLRequestNullTokenString__"
    17 => 2
    18 => "GET"
    19 => {
      "__hhaa__" => "

YnBsaXN0MDDTAQIDBAYIXxAPQWNjZXB0LUVuY29kaW5nVkFjY2VwdF8QD0FjY2VwdC1
MYW5ndWFnZaEFXxARZ3ppcCwgZGVmbGF0ZSwgYnKhB1MqLyqhCVVqYS1qcAgPISg6PF
BSVlgAAAAAAAABAQAAAAAAAAAKAAAAAAAAAAAAAAAAAAAAAXg=="
      "Accept" => "*/*"
      "Accept-Encoding" => "gzip, deflate, br"
      "Accept-Language" => "ja-jp"
    }
    20 => "__CFURLRequestNullTokenString__"
    21 => "__CFURLRequestNullTokenString__"
  ]
  "Version" => 9
}
```

図-3 request_objectのデータのパース結果

```
% echo
YnBsaXN0MDDTAQIDBAYIXxAPQWNjZXB0LUVuY29kaW5nVkFjY2VwdF8QD0FjY2VwdC1
MYW5ndWFnZaEFXxARZ3ppcCwgZGVmbGF0ZSwgYnKhB1MqLyqhCVVqYS1qcAgPISg6PF
BSVlgAAAAAAAABAQAAAAAAAAAKAAAAAAAAAAAAAAAAAAAAAXg== | base64 -d |
plutil -p -
{
  "Accept" => [
    0 => "*/*"
  ]
  "Accept-Encoding" => [
    0 => "gzip, deflate, br"
  ]
  "Accept-Language" => [
    0 => "ja-jp"
  ]
}
```

図-4 request_objectの__hhaa__のパース結果

```
% plutil -p cfurl_cache_blob_data__response_object_2.bin
{
  "Array" => [
    0 => {
      "_CFURLString" => "https://www.example.com/"
      "_CFURLStringType" => 15
    }
    1 => 628074307.809312
    2 => 0
    3 => 200
    4 => {
      "__hhaa__" => "

YnBsaXN0MDDdAQIDBAUGBwgJCgsMDQ4QEhQWGBocHiAiJCZcQ29udGVudC1UeXBlVEV
0YWdXWC1DYWNoZVNBZ2VfEBBDb250ZW50LUVuY29kaW5nVlNlcnZlcldFeHBpcmVzXU
NhY2hlLUNvbnRyb2xURGF0ZV5Db250ZW50LUxlbmd0aF1BY2NlcHQtUmFuZ2VzVFZhc
nldTGFzdC1Nb2RpZmllZKEPXxAYdGV4dC9odG1sOyBjaGFyc2V0PVVURi04oRFcIjMx
NDc1MjY5NDcioRNTSElUoRVWNTY4Nzk3oRdUZ3ppcKEZXkVDUyAobnliLzFEMkYpoRt
fEB1UaHUsIDAzIERlYyAyMDIwIDA5OjA1OjA3IEdNVKEdXm1heC1hZ2U9NjA0ODAwoR
9fEB1UaHUsIDI2IE5vdiAyMDIwIDA5OjA1OjA3IEdNVKEhUzY0OKEjVWJ5dGVzoSVfE
A9BY2NlcHQtRW5jb2RpbmehJ18QHVRodSwgMTcgT2N0IDIwMTkgMDc6MTg6MjYgR01U
AAgAIwAwADUAPQBBAFQAWwBjAHEAdgCFAJMAmACmAKgAwwDFANIA1ADYANoA4QDjAOg
A6gD5APsBGwEdASwBLgFOAVABVAFWAVwBXgFwAXIAAAAAAAACAQAAAAAAAAoAAAAAA
AAAAAAAAAAAAABkg=="
      "Accept-Ranges" => "bytes"
      "Age" => "568797"
      "Cache-Control" => "max-age=604800"
      "Content-Encoding" => "gzip"
      "Content-Length" => "648"
      "Content-Type" => "text/html; charset=UTF-8"
      "Date" => "Thu, 26 Nov 2020 09:05:07 GMT"
      "Etag" => ""3147526947""
      "Expires" => "Thu, 03 Dec 2020 09:05:07 GMT"
      "Last-Modified" => "Thu, 17 Oct 2019 07:18:26 GMT"
      "Server" => "ECS (nyb/1D2F)"
      "Vary" => "Accept-Encoding"
      "X-Cache" => "HIT"
    }
    5 => 1256
    6 => "text/html"
  ]
  "Version" => 1
}
```

図-5 response_objectのデータのパース結果

*3 DB Browser for SQLite(https://sqlitebrowser.org/)。

cfurl_cache_receiver_dataテーブル(表-2)のreceiver_dataにはサーバからのレスポンスボディが保存されています。この情報もフォレンジック解析に有用であるため、解析結果に保存します。ただし、fsCacheDataディレクトリ以下にファイルとして保存されているデータは解析結果に含めるのではなく、ファイルのエクスポートをするのが良いでしょう。これにより、必要以上に解析結果のファイルサイズを大きくさせずに済みます。

2.2.2 データ取得方法

次にデータの取得方法を検討します。Cache.dbはSQLiteデータベースであるため、SQLクエリで情報を取り出すことができます。上記で確認した結果から、必要な情報を取得するためのSQLクエリは図-6のようになります。

response_objectとrequest_objectに保存されているplistデータは、Pythonのplistlibモジュールを使ってパースすることができます[*4]。__hhaa__はパース結果から除外します。

2.2.3 プラグインの実装方針

取得するデータと取得方法について整理できましたので、これらを基にプラグインの実装方針を以下のように決めました。

1. macOSのディスクイメージまたはエクスポート済みのアーティファクトに対して処理を行う。
   1.1 ディスクイメージを処理する場合、全ユーザの「~/Library/Caches/」以下のすべてのアーティファクトを処理する。
   1.2 エクスポートされたアーティファクトを処理する場合、指定されたディレクトリ以下のすべてのアーティファクトを処理する。
2. 解析結果として、以下のデータを保存する。
   2.1 図-6のSQLクエリで得られるデータ
   2.2 response_objectとrequest_objectのパース結果
   2.3 Application Bundle ID
   2.4 ディスクイメージの場合、fsCacheDataディレクトリ以下のファイルをエクスポートする

```
SELECT entry_ID, time_stamp, request_key, request_object, response_object, isDataOnFS, receiver_data
       FROM cfurl_cache_response JOIN cfurl_cache_blob_data USING (entry_ID)
       JOIN cfurl_cache_receiver_data USING (entry_ID)
```

図-6 Cache.dbから必要なデータを取得するSQLクエリ

＊4　mac_aptが提供しているCommonFunctions.ReadPlist()メソッドを使用することもできます。

2.3 プラグインの作成

それでは、プラグインの作成に入ります。今回、解説するプラグインは2021年7月にPull Requestを行い、マージされていますので、mac_aptのGitHubリポジトリで確認することができます[*5]。

なお、本稿ではプラグインの処理の概要を解説します。詳細については、必要に応じてソースコードを確認してください[*6]。

2.3.1 プロパティ

プロパティは図-7のように設定しました。プラグインは「CFURLCACHE」としています。また、ディスクイメージとエクスポート済みのアーティファクトを処理するため、「__Plugin_Modes = "MACOS,ARTIFACTONLY"」としています。

2.3.2 エントリーポイント

■ Plugin_Start()

Plugin_Start()はmac_apt.pyを実行した場合のプラグインのエントリーポイントです(図-8)。

「mac_info.users」はディスクイメージ内のユーザ情報がlistオブジェクトで保存されています。これを利用して、全ユーザのアーティファクトを処理することができます(179行目)。ただし、複数のアカウントに同じホームディレクトリが設定さ

```
__Plugin_Name = "CFURLCACHE" # Cannot have spaces, and must be all caps!
__Plugin_Friendly_Name = "CFURL cache"
__Plugin_Version = "1.0"
__Plugin_Description = "Parses CFURL cache and extract date, URL, request, response, and received data."
__Plugin_Author = "Minoru Kobayashi"
__Plugin_Author_Email = "unknownbit@gmail.com"

__Plugin_Modes = "MACOS,ARTIFACTONLY" # Valid values are 'MACOS', 'IOS, 'ARTIFACTONLY'
__Plugin_ArtifactOnly_Usage = 'Provide the path to "/Library/Cache/" folder under user home'
```

図-7 プロパティ

```
173 def Plugin_Start(mac_info):
174     '''Main Entry point function for plugin'''
175     cfurl_cache_artifacts = []
176     cfurl_cache_base_path = '{}/Library/Caches/'
177     processed_paths = set()
178
179     for user in mac_info.users:
180         if user.home_dir in processed_paths:
181             continue # Avoid processing same folder twice (some users have same folder! (Eg: root & daemon))
182         processed_paths.add(user.home_dir)
183         base_path = cfurl_cache_base_path.format(user.home_dir)
184         if not mac_info.IsValidFolderPath(base_path):
185             continue
186         cache_folder_list = mac_info.ListItemsInFolder(base_path, EntryType.FOLDERS, include_dates=False)
187         app_bundle_ids = [folder_item['name'] for folder_item in cache_folder_list]
188         for app_bundle_id in app_bundle_ids:
189             cache_folder_path = os.path.join(cfurl_cache_base_path.format(user.home_dir), app_bundle_id)
190             cache_db_path = os.path.join(cache_folder_path, 'Cache.db')
191             if mac_info.IsValidFilePath(cache_db_path) and mac_info.GetFileSize(cache_db_path) > 0:
192                 ExtractAndReadCFURLCache(mac_info, cfurl_cache_artifacts, user.user_name, app_bundle_id, cache_folder_path)
193
194     if len(cfurl_cache_artifacts) > 0:
195         PrintAll(cfurl_cache_artifacts, mac_info.output_params, '')
196     else:
197         log.info('No CFURL cache artifacts were found!')
```

図-8 Plugin_Start()関数

*5 cfurl_cache.py(https://github.com/ydkhatri/mac_apt/blob/3e823ee36bdf133c4de3503848435033ee20943d/plugins/cfurl_cache.py)。
*6 プラグインのアップデートが行われた場合、ソースコードと記事内の行数が一致しない場合があります。

れている場合があるため、一度処理したホームディレクトリはスキップします(180～182行目)。

未処理のホームディレクトリの場合、mac_infoオブジェクトのIsValidFolderPath()メソッドでディスクイメージ内のホームディレクトリの存在を確認し(184行目)、ListItemsInFolder()メソッドで「~/Library/Caches/」以下のディレクトリをリストアップします(186行目)。

その後、IsValidFilePath()メソッドで各ディレクトリ以下のCache.dbファイルの存在を確認します(191行目)。存在する場合、ExtractAndReadCFURLCache()関数を呼び出し、Cache.dbの解析結果の取得及びアーティファクトファイルのエクスポートを行います(192行目)。解析結果はcfurl_cache_artifactsにlistオブジェクトとして保存されます。

すべてのCache.dbの解析が終わったら、PrintAll()関数で解析結果を保存します(195行目)。

■ Plugin_Start_Standalone()

Plugin_Start_Standalone()はmac_apt_artifact_only.pyを実行した場合のプラグインのエントリーポイントです(図-9)。

input_files_listにコマンドラインで指定された処理対象のディレクトリ名が入ります(202行目)。その後の処理の流れは、基本的にPlugin_Start()関数と同じになりますが、このエントリーポイントでは、ExtractAndReadCFURLCache()関数ではなく、OpenAndReadCFURLCache()関数を呼び出して解析を行います(212行目)。

2.3.3 データ解析

■ ExtractAndReadCFURLCache()

ディスクイメージ内のCache.dbファイルを開いて、データの解析を行い、解析結果の保存とアーティファクトファイルのエクスポートを行う関数です(図-10)。

```
199  def Plugin_Start_Standalone(input_files_list, output_params):
200      '''Main entry point function when used on single artifacts (mac_apt_singleplugin), not on a full disk image'''
201      log.info("Module Started as standalone")
202      for input_path in input_files_list:
203          log.debug("Input file passed was: " + input_path)
204          cfurl_cache_artifacts = []
205          if os.path.isdir(input_path):
206              cache_folder_list = os.listdir(input_path)
207              app_bundle_ids = [f for f in cache_folder_list if os.path.isdir(os.path.join(input_path, f))]
208              for app_bundle_id in app_bundle_ids:
209                  cache_folder_path = os.path.join(input_path, app_bundle_id)
210                  cache_db_path = os.path.join(cache_folder_path, 'Cache.db')
211                  if os.path.isfile(cache_db_path) and os.path.getsize(cache_db_path) > 0:
212                      OpenAndReadCFURLCache(cfurl_cache_artifacts, '', app_bundle_id, cache_folder_path)
213
214          if len(cfurl_cache_artifacts) > 0:
215              PrintAll(cfurl_cache_artifacts, output_params, input_path)
216          else:
217              log.info('No CFURL cache artifacts were found!')
```

図-9 Plugin_Start_Standalone()関数

```
145  def ExtractAndReadCFURLCache(mac_info, cfurl_cache_artifacts, username, app_bundle_id, folder_path):
146      cfurl_cache_db_path = os.path.join(folder_path, 'Cache.db')
147      db, wrapper = OpenDbFromImage(mac_info, cfurl_cache_db_path, username)
148      if db:
149          ParseCFURLEntry(db, cfurl_cache_artifacts, username, app_bundle_id, cfurl_cache_db_path)
150          mac_info.ExportFolder(folder_path, os.path.join(__Plugin_Name, username), True)
151          db.close()
```

図-10 ExtractAndReadCFURLCache()関数

Cache.dbはOpenDbFromImage()関数内でオープンします(147行目)。この関数ではmac_aptが提供するSqliteWrapperクラスのconnect()メソッドを使用して、SQLiteデータベースのコネクションを取得します。このクラスはPython標準のsqlite3モジュールのラッパーであるため、sqlite3のメソッドを使ってSQLクエリの実行などを行うことができます。

なお、データ解析はExtractAndReadCFURLCache()関数内では行わず、ParseCFURLEntry()関数で行います。これは後述するOpenAndReadCFURLCache()関数と処理を共通化させるためです(149行目)。

最後に、ExportFolder()メソッドでアーティファクトファイルをエクスポートします(150行目)。第1引数はエクスポートを行うフォルダパス、第2引数はエクスポート先のフォルダ名、そして、第3引数は上書きのフラグです。

第2引数で指定したエクスポート先のフォルダは、mac_aptのコマンドラインで指定した出力先のフォルダ以下に作られる「Export」フォルダの下に作られます。他のプラグインのソースコードを確認すると、基本的に「__Plugin_Name」が指定されています。しかし、CFURL Cacheはユーザごとにアーティファクトファイルが存在するため、エクスポート先のフォルダ名としてユーザ名も含めるようにしています。

■ OpenAndReadCFURLCache()

エクスポート済みのCache.dbファイルを開いて、データの解析を行い、解析結果の保存を行う関数です(図-11)。アーティファクトファイルのエクスポートは行いません。

Cache.dbはOpenDb()関数内でオープンします。この関数では、mac_aptのCommonFunctionsクラスのopen_sqlite_db_readonly()メソッドでコネクションを取得します。データ解析は上述したようにParseCFURLEntry()関数で行います。

■ ParseCFURLEntry()

SQLクエリを発行して、必要なデータをCache.dbから取得する関数です(図-12)。また、取得したデータのパースも行い、解析結果を保存します。

最初にテーブル名のリストを取得し、cfurl_cache_schema_versionテーブルがある場合、スキーマバージョンを取得します(118～121行目)。筆者が確認した範囲ではバージョンは「202」しか確認できませんでした[*7]。

次に図-6のSQLクエリを使って、必要なデータを取得します(125～128行目)。なお、上記したように、request_object及びresponse_objectのデータはバイナリフォーマットのplistです。これらのデータは後述するParseRequestObject()

```
153  def OpenAndReadCFURLCache(cfurl_cache_artifacts, username, app_bundle_id, folder_path):
154      cfurl_cache_db_path = os.path.join(folder_path, 'Cache.db')
155      db = OpenDb(cfurl_cache_db_path)
156      if db:
157          ParseCFURLEntry(db, cfurl_cache_artifacts, 'N/A', app_bundle_id, cfurl_cache_db_path)
158          db.close()
```

図-11 OpenAndReadCFURLCache()関数

*7　macOSのバージョンによっては、cfurl_cache_schema_versionテーブル自体がない場合もありました。

関数とParseResponseObject()関数で解析します(130～131行目)。

receiver_dataのオブジェクトの型は保存されている内容によって異なります。レスポンスボディが保存されている場合は「bytes」となり、fsCacheDataディレクトリ以下のファイル名(UUID)が保存されている場合は「str」になります(132～135行目)。

最後に、取得したデータを解析結果のエントリーとして保存します(140～143行目)。解析結果の保存先はCfurlCacheItemクラスとして定義されており(図-13)、そのインスタンスがエントリーとなります。このクラスにはメソッドは定義されておらず、単にデータをひとまとめにするためだけの役割です。

```
114  def ParseCFURLEntry(db, cfurl_cache_artifacts, username, app_bundle_id, cfurl_cache_db_path):
115      db.row_factory = sqlite3.Row
116      tables = CommonFunctions.GetTableNames(db)
117      schema_version = 0
118      if 'cfurl_cache_schema_version' in tables:
119          schema_version = CheckSchemaVersion(db)
120      else:
121          log.debug('There is no cfurl_cache_schema_version table.')
122
123      if 'cfurl_cache_response' in tables:
124          if schema_version in (0, 202):
125              query = """SELECT entry_ID, time_stamp, request_key, request_object, response_object, isDataOnFS, receiver_data
126                         FROM cfurl_cache_response JOIN cfurl_cache_blob_data USING (entry_ID)
127                         JOIN cfurl_cache_receiver_data USING (entry_ID)"""
128              cursor = db.execute(query)
129              for row in cursor:
130                  http_req_method, req_headers = ParseRequestObject(row['request_object'])
131                  http_status, resp_headers = ParseResponseObject(row['response_object'])
132                  if type(row['receiver_data']) == bytes:
133                      received_data = row['receiver_data']
134                  elif type(row['receiver_data']) == str:
135                      received_data = row['receiver_data'].encode()
136                  else:
137                      log.error('Unknown type of "receiver_data": {}'.format(type(row['receiver_data'])))
138                      continue
139
140                  item = CfurlCacheItem(row['time_stamp'], row['request_key'], http_req_method, req_headers,
141                                        http_status, resp_headers, row['isDataOnFS'], received_data,
142                                        username, app_bundle_id, cfurl_cache_db_path)
143                  cfurl_cache_artifacts.append(item)
```

図-12 ParseCFURLEntry()関数

```
41  class CfurlCacheItem:
42      def __init__(self, date, url, method, req_header, http_status, resp_header, isDataOnFS, received_data, username, app_bundle_id, source):
43          self.date = date
44          self.url = url
45          self.method = method
46          self.req_header = req_header
47          self.http_status = http_status
48          self.resp_header = resp_header
49          self.isDataOnFS = isDataOnFS
50          self.received_data = received_data
51          self.username = username
52          self.app_bundle_id = app_bundle_id
53          self.source = source
```

図-13 解析結果を保存するクラス

■ ParseRequestObject()及びParseResponseObject()

request_object及びresponse_objectからデータを取得する関数です(図-14、図-15)。

上記の通り、request_objectのArrayの18番のデータはHTTPメソッド、19番のデータはHTTPリクエストヘッダになります(図-14:96 ～97行目)。また、__hhaa__は除外します(図-14:100行目)。同様に、response_objectのArrayの3番はHTTPステータス、4番目はHTTPレスポンスヘッダになります(図-15:106 ～107行目)。

2.3.4 解析結果保存

■ PrintAll()

解析結果をコマンドラインで指定されたフォーマットで保存する関数です(図-16)。cfurl_cache_infoは解析結果を保存する際のカラム名とその型を定義しています(162 ～164行目)。この定義と同じ順番で解析データをlistオブジェクトとしてまとめ、最後にWriteList()関数でファイルへの書き込みを行います(168 ～171行目)。

2.4 プラグイン実行例

今回解説したプラグインによる解析結果は図-17のようになります(Received_Dataカラムより右側はスクリーンショットからカットしています)。情報が整理され、データを確認しやすくなったのではないかと思います。

```
94  def ParseRequestObject(object_data):
95      object_array = plistlib.loads(object_data)['Array']
96      http_req_method = object_array[18]
97      header_list = object_array[19]
98      req_headers = []
99      for header, value in header_list.items():
100         if header != '__hhaa__':
101             req_headers.append("{}: {}".format(header, value))
102     return http_req_method, "\r\n".join(req_headers)
```

図-14 ParseRequestObject()関数

```
104 def ParseResponseObject(object_data):
105     object_array = plistlib.loads(object_data)['Array']
106     http_status = object_array[3]
107     header_list = object_array[4]
108     resp_headers = []
109     for header, value in header_list.items():
110         if header != '__hhaa__':
111             resp_headers.append("{}: {}".format(header, value))
112     return http_status, "\r\n".join(resp_headers)
```

図-15 ParseResponseObject()関数

```
161 def PrintAll(cfurl_cache_artifacts, output_params, source_path):
162     cfurl_cache_info = [('Date', DataType.TEXT), ('URL', DataType.TEXT), ('Method', DataType.TEXT), ('Request_Header', DataType.TEXT),
163                         ('HTTP_Status', DataType.TEXT), ('Response_Header', DataType.TEXT), ('isDataOnFS', DataType.INTEGER), ('Received_Data', DataType.BLOB),
164                         ('User', DataType.TEXT), ('App_Bundle_ID', DataType.TEXT), ('Source', DataType.TEXT)]
165
166     data_list = []
167     log.info(f"{len(cfurl_cache_artifacts)} CFURL cache artifact(s) found")
168     for item in cfurl_cache_artifacts:
169         data_list.append([item.date, item.url, item.method, item.req_header, item.http_status, item.resp_header, item.isDataOnFS, item.received_data, item.username, item.
            app_bundle_id, item.source])
170
171     WriteList("CFURL cache", "CFURL_Cache", data_list, cfurl_cache_info, output_params, source_path)
```

図-16 PrintAll()関数

	Date	URL	Method	Request_Header	HTTP_Status	Response_Header	isDataOnFS	Received_Data
	フィルター	フィルター	フィル...	フィルター	フィルター	フィルター	フィルター	フィルター
1	2020-11-09 01:58:45	https://stackoverflow.com/	GET	Accept: */*...	200	Content-Encoding: gzip...	1	2B1680C0-DAE0-4EA0-9EC0-C4FC7F86A8C0
2	2020-11-09 02:10:43	https://www.example.com/	GET	Accept: */*...	200	Content-Type: text/html; charset=UTF-8...	0	<!doctype html>...
3	2021-02-02 06:58:03	https://raw.githubusercontent.com/its-a-feature/Orchard/master/Orchard.js	GET	Accept: */*...	200	Content-Encoding: gzip...	1	A391D5EC-9FCF-4993-A0AF-EEF2C871EF6A

図-17 プラグインによる解析結果

2.5 まとめ

前回から2号にわたって、mac_aptプラグインの作成について解説を行いました。おおよそのプラグインの構造と処理の流れは理解していただけたのではないかと思いますが、mac_aptが提供しているすべてのAPIを解説したわけではありません。他のプラグインも参考にすれば、より理解が深まるのではないかと思います。

mac_aptは強力なフォレンジック解析ツールですが、より多くのアーティファクトをサポートするにはユーザ自身がプラグインを作成するのが一番です。前号で解説したように多くのアーティファクトは、plistまたはSQLiteフォーマットであるため、データを参照すること自体は非常に簡単ですし、何より、自分が必要とするデータを好みの形式で解析することができるというメリットがあります。

なお、プラグインの作成よりも、mac_aptの解析結果をいかに読み解けば良いかという点に興味がある方もいるのではないかと思います。そのような方には、Japan Security Analyst Conference 2022(JSAC2022)で開催されたmacOSフォレンジックハンズオンワークショップで使用した資料[*8]と解析データ[*9]が参考になるのではないかと思います。このワークショップでは、mac_aptの解析結果を用いて、マルウェア侵害のフォレンジックタイムラインを作成します。動画もアーカイブとして公開されています[*10]。

執筆者:
小林 稔(こばやし みのる)

IIJセキュリティ本部セキュリティ情報統括室フォレンジックインベスティゲーター。
IIJ-SECTメンバーで主にデジタルフォレンジックを担当し、インシデントレスポンスや社内の技術力向上に努める。
Black HatやFIRST TC、JSAC、セキュリティキャンプなどの国内外のセキュリティ関連イベントで講演やトレーニングを行う。

*8 ワークショップ資料(https://jsac.jpcert.or.jp/archive/2022/pdf/JSAC2022_workshop_macOS-forensic_jp.pdf)。
*9 解析データ(https://jsac.jpcert.or.jp/archive/2022/data/JSAC2022_macos_forensic_workshop_without_malware.7z)。
*10 [JSAC2022] Workshop: An Introduction to macOS Forensics with Open Source Software(https://www.youtube.com/watch?v=Mor9EplnrXM)。

サービス基盤でのストレージに関する課題と対応

3.1 はじめに

IIJでは、2000年にクラウドサービス「IIJ GIO」の前身であるリソースオンデマンドサービス「IBPS」を開始した当時からサービス基盤にはストレージアレイシステムを採用しています。そして現在もIIJ GIOと自社サービス向けクラウド「次世代ホストネットワーク:NHN(Next Host Network)」においては、それぞれの基盤にストレージアレイシステムを採用しており、ストレージの容量は日々増え続けています。

本稿では、まずストレージについて理解を深めていただくために一般的なストレージについて紹介します。そして、お客様に安心してサービスをご利用いただくための取り組みとして、IIJがどのようなストレージ及びストレージネットワークを採用しているのかについて紹介します。

3.2 ストレージの一般的な話

3.2.1 ストレージとは?

ストレージとは、データやプログラムを保管するものの総称です。身近なコンシューマ向けストレージの例は、パソコンに入っているハードディスクドライブ(HDD)やソリッド・ステート・ドライブ(SSD)、データを持ち運ぶときに使うUSBメモリ、スマートフォンやデジタルカメラに挿入されているSDカード、CDやBlu-rayなど映像・音楽で使用されるメディア、ネットワーク対応HDDと呼ばれるNASなどが挙げられます。パソコンなどを長年使用し続けていれば、一度はHDDやSSDの故障によりパソコンが使えなくなった経験があるのではないでしょうか。

IIJのサービス基盤でも採用しているエンタープライズ向けストレージは、故障でストレージが止まることがないように、HDDやSSD、その他部品が複数搭載され、どれか1つ故障したとしてもデータの消失やストレージへのアクセスが停止しない可用性の高い構成となっています。このような構成のストレージを、「ストレージアレイシステム」や「ディスクアレイシステム」「エンタープライズストレージ」と呼んだりしますが、通常我々は単純に「ストレージ」と呼んでいます。

この節では、ストレージの部品として使用されるHDDとSSDに的を絞り、ストレージアレイシステムの仕組みや、ストレージアレイシステムとサーバを繋ぐネットワークについて解説します。

3.2.2 HDDとSSDの違い

HDDやSSDは、どちらもブロックストレージに分類されるストレージで、通常PCやサーバで利用する場合、インストールされているOSで使用するファイルシステムでフォーマットした後、利用するのが一般的です。

HDDは、プラッタと呼ばれる磁性体が塗布された円盤を高速回転させ、磁気ヘッドからプラッタにデータを読み書きします。プラッタと磁気ヘッドの隙間は人間の髪の毛の太さよりも狭いことから衝撃に弱く、動作中にHDDを移動させると故障することもあります。また、モーターなど機械的に駆動する部分があるため、これら駆動部分の動作不良もドライブ故障の要因となります。

HDDの速度はプラッタの回転数により決まります。ご存じのとおり、つい数年前まで販売されていたパソコンはストレージにHDDが使用されていたため、パソコンの電源を入れてWindowsなどのOSが起動するまで長く待たされるのが普通でした。このようにHDDは昨今のシステムにおいてボトルネックとなることがあります。この速度を高速化する製品として、最近はSSDを使用することが多くなりました。

SSDはフラッシュメモリと呼ばれる半導体素子を用いており、HDDと比較すると機械的な駆動部分がないため静かで衝撃に強く、発熱も比較的少なく、何よりHDDに比べると読み書き速度が高速です。SSDはHDDに比べると故障は少ない傾向ですが、使用しているフラッシュメモリの特徴として書き換え回数による寿命が存在します。

HDD及びSSDのコンシューマ向けとエンタープライズ向けの違いですが、エンタープライズ向け製品は、部品の精度や耐久性、寿命といった点でコンシューマ向けより上であるといえ

ます。またエンタープライズ向けは出荷前にエージングを行って初期不良を排除するような製品もあるようです。

3.2.3 RAID

HDDやSSDに限った話ではありませんが、どちらも故障の要因が存在している以上、故障に伴うデータロストの可能性があります。このデータロストを防ぐ方法として、一般的なストレージ装置ではRAID(Redundant Array of Independent Disks)という手法を用いて、データの可用性を高めています。RAIDにはいくつかの種類があり、現在の主流は「RAID 1」、「RAID 5」、「RAID 6」です。

RAID 1は最低2台のドライブにデータをミラーリングし、どちらかのドライブに故障が発生してもデータを保護できます。RAID 5は最低3台のドライブを使い、パリティ(誤り訂正符号)をそれぞれのドライブに分散記録します。1台のドライブ故障までデータを保護することができ、2台壊れるとデータ破損が発生します。RAID 6は最低4台のドライブを使い、2種類のパリティをそれぞれのドライブに分散記録します。2台のドライブ故障までデータを保護することができ、3台壊れるとデータに破損が発生します。

余談ですが、データ保護の目的ではなくパフォーマンスを上げる目的の「RAID 0」があり、通常はRAID 0単体で使用することはなく、前述のRAID 1、RAID 5、RAID 6と組み合わせて使います。

RAIDは作成されたRAIDの領域から一部分、もしくは全体を使用してLogical Unit(LU)やVolumeを作成し、サーバにディスクデバイスとして提供するのが一般的です。

またRAIDはスペアドライブという機能を持っており、通常は未使用のドライブですが、ドライブ故障が発生したら自動的にデータを退避させる、もしくは残っているドライブとパリティから計算しデータをスペアドライブに復元させます。RAIDのスペアドライブ機能はずいぶん前から一般的に利用されている技術ですが、この構成の欠点として、ドライブ故障時にスペアドライブにI/Oが集中し性能低下が発生してしまいます。そこで、最近のストレージアレイシステムでは、搭載されているドライブすべてにスペアの領域を設け、ドライブ故障時の性能低下を軽減する機能を持つ製品も販売されています。

ここまでRAIDを説明してきましたが、それぞれのRAIDの例として、4TB HDD 6本(スペアドライブなし)で構成した場合、使用できる容量と可用性について表にまとめてみました(表-1)。

RAID 1は一番容量効率が悪く、次いでRAID 6、そしてRAID 5は一番容量効率が良いことが分かります。IIJで使用しているストレージでは、利用効率と可用性を加味し、RAID 5またはRAID 6を採用しています。

3.2.4 ストレージアレイシステム

ストレージアレイシステムは、ストレージの機能を司るコントローラと各種ドライブ・電源ユニットが複数搭載され、どれか1つ故障が発生したとしてもストレージへの読み書きが継続して行えるよう冗長構成になっています。通常ストレージアレイシステムとサーバなどを接続して利用するためには、ネットワークで接続する必要があり、データのやりとり方法の違いによってプロトコルが異なります。具体的には、ファイルアクセスに使用するプロトコル、ブロックアクセスに使用するプロトコル、Amazon AWSのS3などオブジェクトアクセスに使用するプロトコルがあります。このうちファイルアクセスとブロックアクセスに絞って解説します。

RAIDレベル	容量	可用性
RAID0	24TB	×
RAID1	12TB	○
RAID5	20TB	○
RAID6	16TB	○

表-1 各RAIDを構成した場合の容量と可用性

3.2.5 ファイルアクセスのプロトコル

まず、ファイルサーバ(NAS)など良く耳にするストレージについて解説します。NASで使用されるプロトコルの種類としては、UNIX/Linux系OSで使用されるNetwork File System(NFS)や、Windowsなどで使用されるServer Message Block(SMB)及びCommon Internet File System(CIFS)があります。一般的なサーバでファイルストレージを構成した場合、RAIDから切り出したボリュームをフォーマットし、その領域をNFSやSMBの共有設定により他のサーバでマウントして利用します。ファイルストレージの用途としては、Webサーバのコンテンツ格納や、各種アプリケーション間のデータ共有、オフィスでのファイル共有などがあります。

3.2.6 ブロックアクセスのプロトコル

次に、HDDやSSDと同様のブロックストレージをネットワーク経由で利用する方法があり、プロトコルの種類はiSCSIやFibre Channel(FC)、最近登場したNVMe over Fabrics(NVMe-oF)があります。主にブロックストレージとサーバを繋ぐネットワークをStorage Area Network(SAN)と呼び、FCを利用したストレージネットワークをFC-SAN、iSCSIなどイーサネットを利用したストレージネットワークをIP-SANと呼ぶのが一般的です。

iSCSIとFCについて簡単に説明すると、1993年にFCを用いた製品の販売が開始され、iSCSIについては2003年にIETFによってRFCとして公開され、その後製品が出荷されました。

iSCSIは、ストレージのプロトコルであるSCSIをイーサネットのプロトコルであるTCP/IP上で使用する規格で、専用のハードウェアを用意する必要はなく、イーサネットスイッチ及びサーバに標準搭載されているNICで構成することが可能です。ストレージ技術に長けている人であれば、ご家庭のネットワークでiSCSIを利用しているかもしれません。

FCは、ストレージのプロトコルであるSCSIを、ストレージ専用に設計されたFCの上で使用する規格で、専用のハードウェアが必要となり、スイッチはFCスイッチ、サーバにはFC-HBAが必要です。かつ、これら製品に対する専門知識が必要になります。

3.3 非常に大規模なFC-SAN

ブロックストレージとサーバを接続するプロトコルとして、「IIJ GIOインフラストラクチャーP2」及び「IIJ GIOインフラストラクチャーP2 Gen.2」では、サーバとストレージの接続にFC SANを導入しています。

2節で解説したとおり、FCは専用のハードウェアが必要かつ専門知識が必要となり、一般的にはハードルが高い印象ですが、IIJでは2003年から自社でFC-SANの構築及び運用を行ってきており、FC-SANとIP-SANの性質について理解し、それぞれの環境において適切に選択できるようになりました。

例えば、FC-SANを導入する理由として、FC-SANで使用するスイッチは、IP-SANで実現するには難しい機能を標準搭載しています。

3.3.1 マルチテナンシー

1つのネットワークに複数のシステムを収容する場合、IP-SANで各システム間のセキュリティを担保する方法として、イーサネットのVLANを利用しネットワークを論理的に分断することが一般的かと思います。図-1のように1つのシステムにそれ専用のストレージを接続する場合は特に難しい構成ではありません。しかし、図-2のように複数のシステムに単一のストレージを接続する場合、ストレージ側のインタフェースで複数のVLAN(VLAN Tagging、IEEE 802.1Q)をサポートする必要が出てきます。

現在販売されているストレージアレイシステムで、VLAN Taggingをサポートしている製品が豊富かというと、NAS製品については多数存在していますが、iSCSIなどIP-SAN向けの製品はごく少数しか存在しておらず、対応していてもVLAN数に制約があり、多数のシステムの収容が困難である場合が少なくありません。

次にFCについて説明すると、FC-SANを流れるプロトコルはイーサネットで使うプロトコルではなくストレージに特化されたプロトコルのみで、FC-SANの目線で見ると各サーバ間のセキュリティは保たれます。また、FCのセキュリティを高める方法として、図-3のようにストレージ(ターゲット)とサーバ(イニシエータ)を紐付けるゾーニングという機能を用いて、他システムが利用するストレージへのアクセスを遮断する方式を取っています。またイーサネットとは異なり、複数システムで単一のストレージを共有する場合は図-4のようにそれぞれのゾーニングを設定すれば良いだけなので、アレイシステムの選択に苦慮しません。他にFC-SANを使うと、サーバとストレージの間のIPアドレス設計が不要になることでしょうか。

また、この例ではFC-SAN、IP-SANとも、スイッチの台数は1台ですが、ストレージとサーバの間に複数台のスイッチがある場合、IP-SANはすべてのスイッチに対してVLANの設定が必要になるのに対し、FC-SANはゾーニング情報が接続されるスイッチすべてで共有される仕組みがあるため、どれか1台だけ設定をすれば良く、作業量を比較するとFC-SANの方が少なくなります。

3.3.2 ロスレスなネットワーク

FCは標準で高性能、低遅延、ロスレス伝送を実現します。イーサネットでは仮にパケットロスが発生した場合、TCPの再送制御を用いてデータの整合性を保っています。このような場合ストレージでパケットロスが頻発する環境下はストレージパフォーマンスの低下に繋がり、アプリケーションの動作に致命的な不具合が発生することもあるため、イーサネットでIP-SANを構成する場合はネットワークの設計などに注意を払う必要があります。

また、FCと同様のロスレスなイーサネットを構成する場合、Data Center Bridging (DCB) などの機能が利用できるスイッチ製品を選択すれば可能となりますが、追加でサーバのNICとストレージにDCBの設定を施すことが必要です。

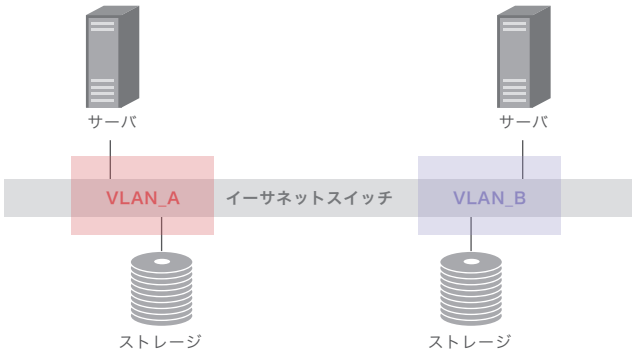

図-1 iSCSIストレージ専用構成

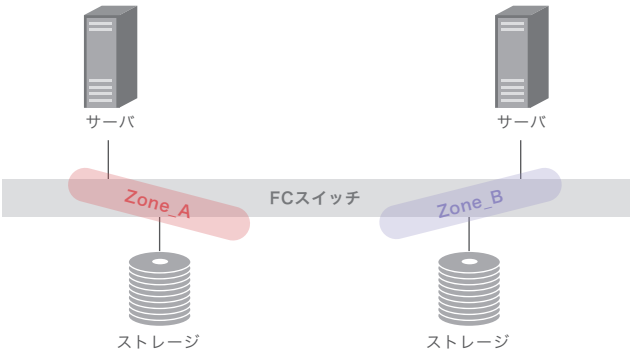

図-3 FCストレージ専用構成

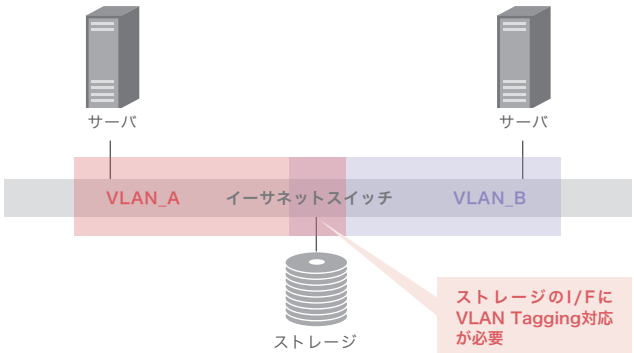

図-2 iSCSIストレージ共用構成

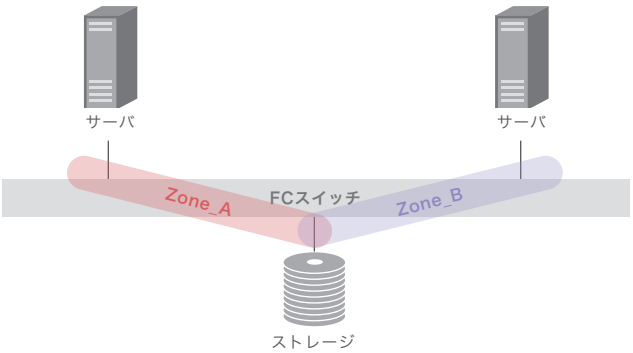

図-4 FCストレージ共用構成

3.3.3 FC Fabric

FC Fabricには以下のようなFabric Topologyを構成することが可能です。図-5ではスイッチ1台でサーバとストレージを繋いでいますが、実際にはスイッチを2セット用意し、サーバとストレージの経路を2経路確保して、単一障害で停止を起こさない構成になっています。

まずはスイッチ1台で構成した場合の構成です。小規模からスタートできるので、まずはこの構成をIIJでも採用していました。この構成の欠点は、サーバやストレージが増えてくると拡張が難しくなり、拡張しようにも大規模なメンテナンスを余儀なくされてしまうことです。

次にCore-Edge、Edge-Core-Edge、Full Meshの構成を図-6に示します。

Core-Edgeは、データセンターの1フロアでストレージ及びサーバが密集している場合においては有効となる構成で、GIO P2の一部、GIO P2 Gen.2で採用しています。図-6ではFCスイッチ間を1本のケーブルで接続しているように見えますが、実際は1本切断しても影響を軽減するために2本以上で接続しています。これは、FCスイッチに備わるInter Switch Link（ISL）という機能を用いており、イーサネットスイッチのPort Channelと似た機能ですが、イーサネットスイッチとは異なりFCスイッチにISL Trunkライセンスが投入されていればひ

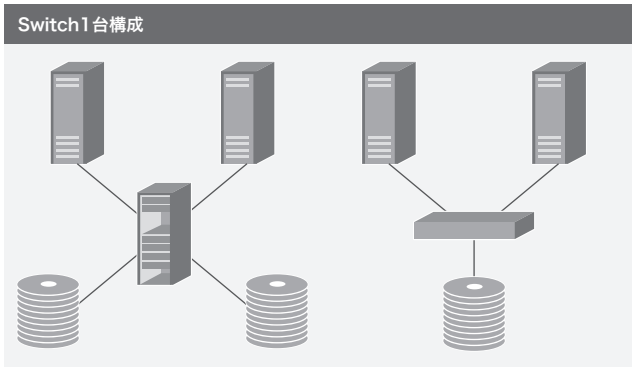

図-5 スイッチ1台構成の場合のFC Fabric Topology

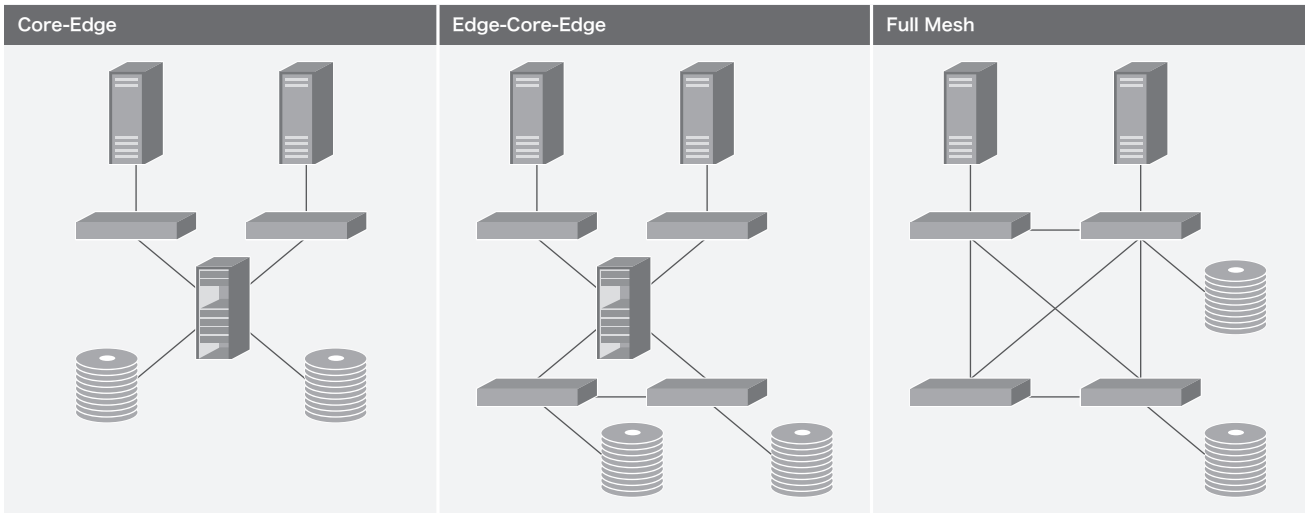

図-6 複数のスイッチを使ったFC Fabric Topology

とまとめのLinkに自動設定されます。例えば32GbpsのFCスイッチ同士で4本をISL Trunkで接続する場合、スイッチ間の帯域は32 x 4 = 128Gbpsとなります。

Edge-Core-Edgeは、データセンターの複数フロアにストレージ及びサーバが分かれて設置されている場合に採用しています。例えば、Coreの近くにストレージが設置できない状況において、ストレージ用Edgeスイッチを配置してCoreと接続する必要があるといった理由によります。また、大規模になるとEdge-Core-Core-EdgeというCoreを別々のフロアに設置する構成もありますが、その場合Core間のトラフィックが多くなると見込めますので、ISL Trunkの本数を予め多めに設計するよう心がけています。

Full Meshは、小規模な構成でデータセンターのラックにストレージ及びサーバが混載されている場合に採用しています。Full Meshの特筆すべき点は、通常のイーサネットスイッチでFull Meshを構成する場合、高度な設定を要する場合が多い点です。FCスイッチでは自動的にストレージ～サーバの経路を適切に設定する機能があるため、この構成にしたとしても特別な設定は不要となり、非常に簡単に構成を組むことが可能です。

3.3.4 FCのファブリックサービス

FCスイッチではファブリックサービスがあり、これはFC-SANに接続されているすべてのデバイスや、サーバとストレージを接続するために必要な情報を管理します。ファブリックサービスのほんの1例を挙げると、ストレージとサーバを繋ぐゾーニングを適切に設定していれば、サーバに特別な設定をせずともストレージと自動的に接続する仕組みです。IP-SANに同様の機能はInternet Storage Name Service(iSNS)として存在はしていますが実装は限定的で利用されている環境はほぼありません。そのため、IP-SANではサーバに接続対象のストレージを記憶させて接続させる方法を取ります。違いをまとめたのが図-7です。

例えばサーバ台数が数台程度であるなら、FC-SANもIP-SANも接続するためのオペレーションの量に大差はないと思いますが、サーバ台数が数十台～100台のような環境の場合、すべてのサーバに接続対象のストレージを記憶させる必要があるため、サーバ

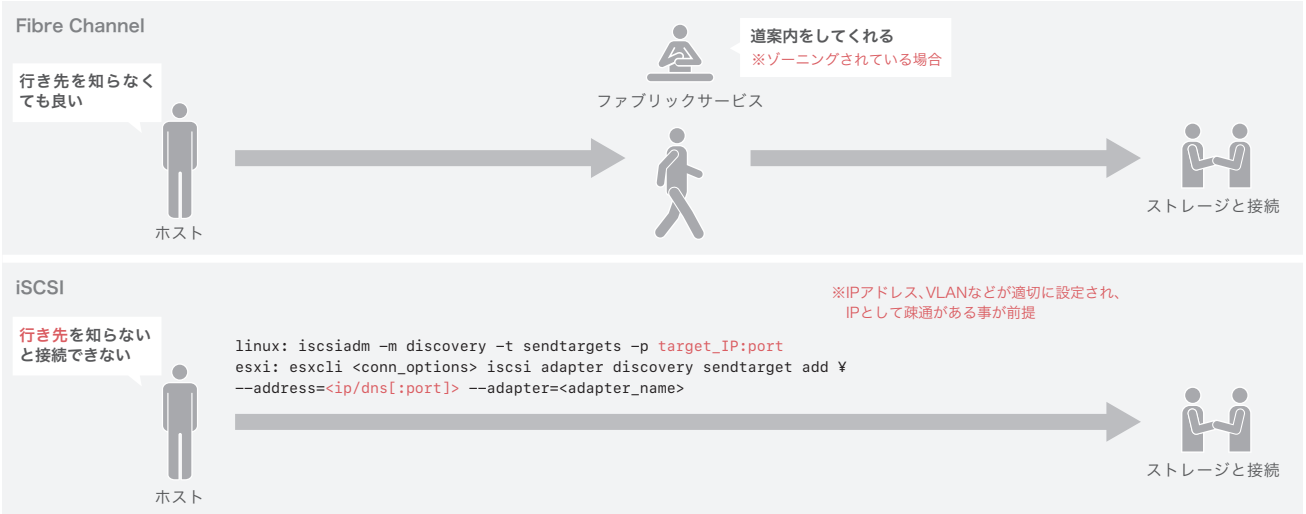

図-7 FC-SANとIP-SANの接続形態の違い

を管理しているエンジニアの作業量は多くなる傾向です。

3.3.5 FC-SANを選択する判断基準

運用面では、FCスイッチのファームウェアバージョンアップ時にFCのトラフィックを止めずにバージョンアップができる機能や、スイッチの各PortでSFP+モジュールの障害やケーブルの劣化によるエラーカウントが一定数超えた場合に強制的にPortをオフラインし、これ以上FC Fabricに悪影響を与えないようにする機能があります。

これらを鑑みて、IIJでは現状ストレージアレイシステムを複数のシステムで利用する要件がない場合はFC-SANもしくはIP-SANどちらでも良く、複数のシステムで共用利用をする要件がある場合はFC-SANを選択するのが望ましいという判断基準をもっています。

しかし将来IP-SANで利用するストレージアレイシステムでVLANなどの対応が柔軟になった製品が増えていく場合、100Gbps以上の伝送速度をもつイーサネットを利用したIP-SANに軍配が上がると予想されるため、今後の判断基準は変更になる可能性もあります。

3.4 運用技術について

IIJではストレージを安定運用するために、以下のような取り組みを行っています。

3.4.1 サービス提供の自動化

「IIJ GIO」で、お客様からストレージを利用したいという申し込みがあった場合、2010年頃まではストレージをメインで運用しているエンジニアがストレージ及びFCスイッチの設定変更を行っていました。この頃まではストレージの設定変更を行う業務は多くても週に1回程度と頻度が少なかったため、エンジニアによる手動の設定変更でも十分期限に間に合いました。

しかし、2010年を過ぎると週に1回の業務が週に数回、更には1日に数回発生するようになり、人手を介して作業するのが困難になると予想されたため、ストレージの設定変更をすべて自動化するようなシステムを設計・構築しました。

ストレージ設定の自動化を行うためには、市販のアプリケーションなどを利用すれば簡単なのですが、市販のアプリケーションはストレージの自動化だけでなく、それ以上の機能を有するものが多く（かつ高価なため）、ストレージを制御する部分は自作としました。

ストレージ制御はPythonやRubyなどの言語を使って記述しています。本来、各種言語からストレージの構成変更を行う場合、ストレージ管理の仕様であるStorage Management Initiative - Specification（SMI-S）や独自APIを経由して構成変更を行うことが望ましいのですが、当時使用していたストレージやFCスイッチそれぞれ単体ではSMI-SやAPIに対応しておらず、別途専用の（高価な）アプリケーションが必要であったため、標準でストレージを制御するコマンドラインを使用しました。

コマンドラインの入出力は人間が確認することには長けていますが、各種言語で結果から適切な値を取り込むには非常に扱いづらい形式で出力されているため、作成したプログラムも相当面倒な文字列処理を使う必要がありました。また、使用していた機器の一部は、コマンド及びパラメータに誤りがあってもリターンコードが0（正常終了）を返すため、独自のエラー処理を追加する必要もありました。

最も手間がかかるのは、ストレージアレイシステムやFCスイッチのファームウェアをバージョンアップする際です。バージョンアップ後のコマンドライン出力結果がバージョンアップ前と比較して若干の変更が発生する製品もあり、本番でバージョンアップする前に開発環境で事前のテストを行う必要がありました。

最近のストレージ及びFCスイッチは、構成管理を行うソフトウェアであるAnsibleなどの対応が進んできていることから機器単体でのAPI実装が進み、かつ、Pythonなどでストレージ制御に使えるモジュールを各ストレージベンダーが提供するようになったため、当時と比べると非常に簡単にプログラムを組

むことが可能となっています。

3.4.2 ストレージ基盤の監視

ストレージシステムの監視方法として、ping、syslog、SNMPなどを利用して監視を行うほか、メーカ独自の監視ツールを利用して、メーカから遠隔の監視を行う方法があります。当初はIIJが提供している監視サービスを利用して監視を行っていました。その頃はストレージ1台に接続されるサーバの台数も少ないため、この方法でも障害発生からストレージ運用チームが障害を認知するまで時間はかかりませんでした。

しかし、規模が大きくなると1台のストレージに接続されるサーバの台数も増え、そのストレージに障害が発生すると、1つの障害に対しIIJの監視サービスで検知するアラートの数が数百件を超えることが発生、ストレージ運用チームが障害を認識する時間がかかるようになりました。

そこで、ストレージ運用チームで独自の監視システムを構築し、ストレージシステムのアラートを直接ストレージ運用チームに通知する方式としました。これにより、数十分かかっていた障害検知を数分レベルに短縮することができました。

3.4.3 データ移行

ストレージアレイシステムに搭載されているHDDやSSDは、おおむね5年間ほど連続稼働すると故障の頻度が増えてきます。IIJでは、1台のストレージアレイシステムの利用期間が5年を経過する頃に機器の入れ替えを検討しています。その際に避けて通れないのがデータ移行です。

vSphere環境であれば、Storage vMotionなどの機能を使うことにより少ない影響でデータ移行できますが、これ以外の環境においては何らかの方法を用いてデータ移行を行う必要が発生します。1つはサーバなどで移行元・移行先ストレージをマウントし、ファイル単位で移行する方法があります。移行の流れは図-8のとおりです。WindowsだとDrag-and-Dropやrobocopy、Linuxだとcpやtar、rsyncなどを使います。

この方法は非常に単純でIIJでもよく利用していましたが、常時ストレージに対して書き込みが行われる環境では1回だけの移行ではファイル全体の不整合が発生してしまいます。そこでIIJでファイル単位のコピーを行う際は、複数回にかつ数日に分けて移行する方法をとりました。具体的な方法としてはまず全体をコピーし、その後1日おきに差分でコピー、平行して書き込みが頻繁に行われるディレクトリを特定し、最後に書き込みを止めるためにアプリケーションを停止し、最後に書き込まれたファイルのみをコピーして移行します。

次に、物理サーバなどで論理ボリュームマネージャを使っている場合は、その論理ボリュームマネージャの機能を使って移行します。移行の流れは図-9のとおりです。

論理ボリュームマネージャの知識は必要になりますが、データの移行方法としては前述のファイル単位で移行する方法より

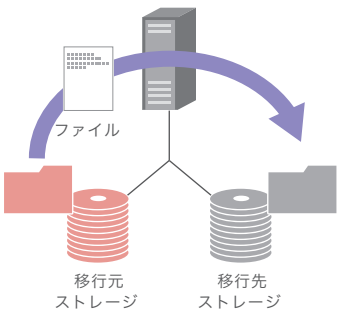

図-8 ファイル単位で移行

も影響が少なく済んでいます。また、ストレージ上で動いているアプリケーションへの影響も、若干の性能低下はあるものの、大幅な性能劣化はありません。具体的な例としては、LinuxやHP-UXで利用している論理ボリューム(LVM)を用いて移行元ストレージと移行先ストレージでミラーを一時的に組んで同期が取れたら移行元ストレージを切り離すという方法や、Linuxに限定するとLVMのpvmoveコマンドを用いて移行する方法などが挙げられます。

ただし、もともと論理ボリュームマネージャを使用していない環境では、この方法を用いることはできないため、そのような環境では前述のファイル単位の移行を行うほかに、後述するストレージの移行機能を用いるしかありません。

最後にストレージの移行機能を使う方法です。移行の流れは図-10のとおりです。

昨今のミッドレンジクラス以上のストレージには、同機種もし

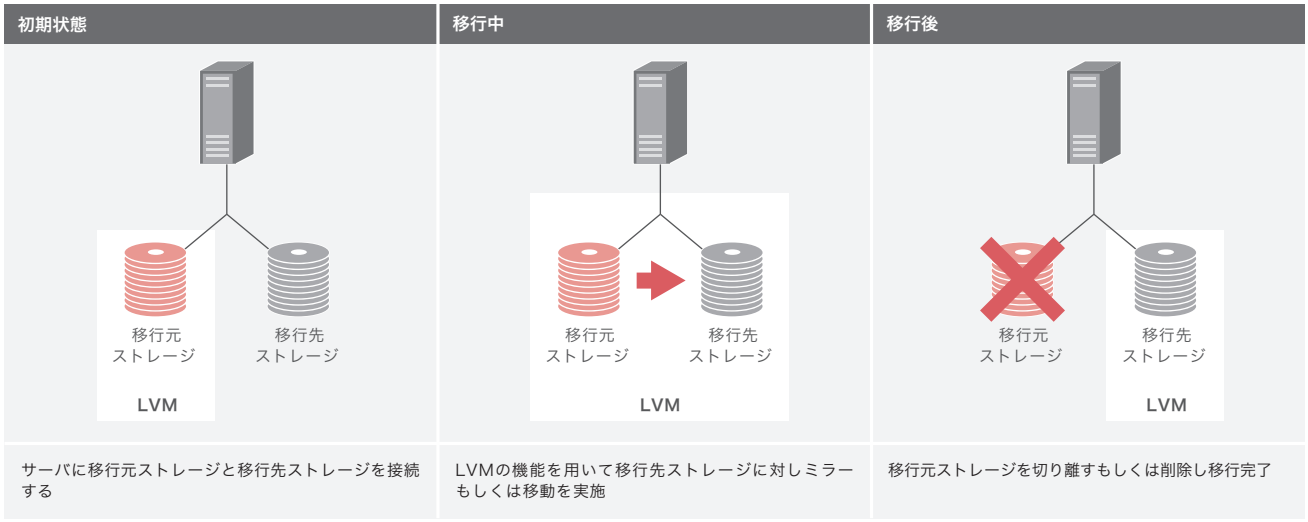

図-9 論理ボリュームの機能で移行

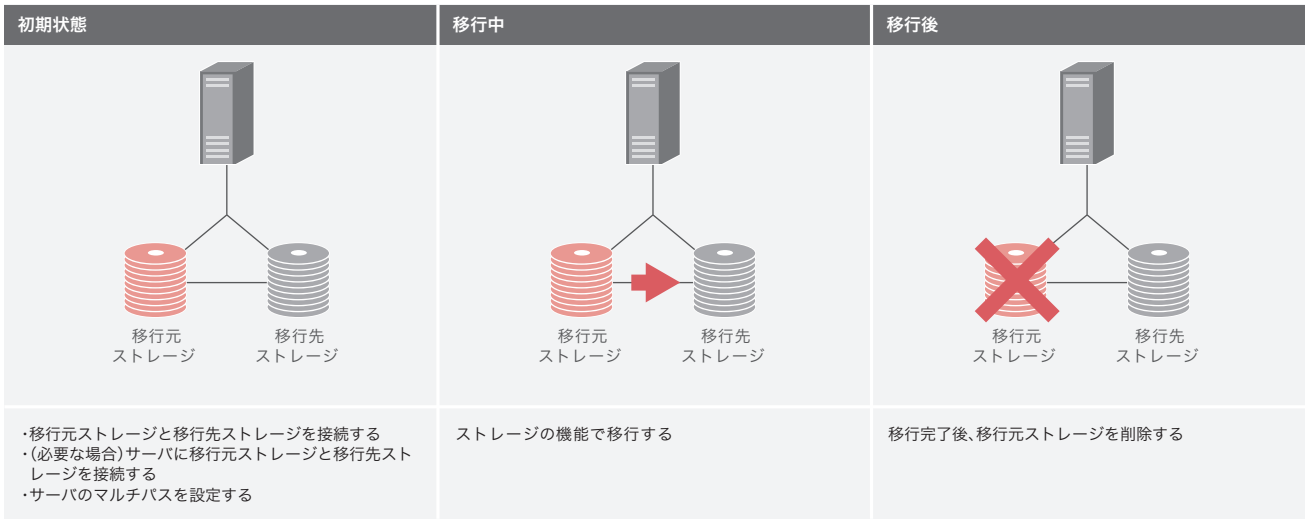

図-10 ストレージの機能で移行

くは同系統のストレージ間でデータ移行ができる機能を備えた製品があり、一部では異なるベンダー間での一方向データ移行を行える製品も出てきました。この方法を使うと、ストレージを使っている環境に左右されず、ほぼすべてのデータ移行を行うことが可能です。ここでの注意点として、例えば移行元ではFCを利用し移行先でiSCSIを利用したいなど、サーバとストレージ間のプロトコルを変えることはできず、移行元と移行先で同じプロトコルを利用することが前提です。また、ストレージを利用している環境によっては、移行作業の前後でサーバの停止を必要とする場合もあります。

3.5 まとめ

本稿では、IIJのストレージの取り組みの紹介として、一般的なストレージ機能、ストレージアレイ及びストレージネットワーク採用の技術説明、ストレージ運用技術について述べました。IIJでは引き続き、お客様にストレージを安心かつ確実にご利用いただけるように努めてまいります。

執筆者:
菊地 孝浩（きくち たかひろ）
IIJ クラウド本部 技術開発室 シニアエンジニア。
入社後、IIJクラウドサービスのストレージ基盤を設計・構築・運用業務に従事。
現在は、ストレージに関する調査や研究活動、社内外の技術者育成活動や、SNIA日本支部の分科会に参加。

Information

IIJ Engineers Blog 最新トピック

■IIJ Engineers Blog 特集「私の愛用している技術系アイテム」

本特集ではIIJのエンジニアが愛用しているアイテムを紹介しています。

キーボードやマウスといった身近なものから、関数電卓や3Dプリンタ、工具、さらにはお守りまで！？個性豊かでバラエティに富んだアイテムが登場します。一口にアイテムといっても、人によって思いは様々――どんなきっかけで出会い、どこが気に入って長年愛用しているのか、アイテムに対する筆者の思いをぜひご覧ください。

[特集一覧]

● 未だに使い続けてるキーボード・マウス	とみ
● プロジェクトリングノート	藤本 椋也
● 大人になったので少し落ち着こうと思いまして（デスクトップPC）	y-morimoto
● ゲーミングマウスは仕事も捗ります	熊本 祐
● PINARELLO FP3（ロードバイク）	田口 景介
● プログラマ電卓 HP-16C	和田 英一
● 社会人人生を共に歩んでいる関数電卓 fx-912s	北河 直樹
● 工具セットと電電宮の御守	竹﨑 友哉

※上記は2022年6月1日現在公開している記事です。本特集は2022年5月～6月にかけて毎週1 ～2本、記事を公開しています。

■動画「インターネット・ミニ解説」

最近の技術動向を分かりやすく解説した動画を公開しています。過去のIIRの解説もありますので、ご興味のある方はぜひご覧ください。

400ギガビットイーサネット
（マルチフィード・IIJ・NTT Com 400Gインターフェース相互接続検証）

（2022年4月19日公開）

クリミア半島のインターネット
（2014年ロシアによるクリミア”併合”の影響）

（2022年3月29日公開）

DDoSとBackscatter

（2022年2月22日公開）

サイバーセキュリティ 2021年12月の動向
「log4j 2」問題について

（2022年1月27日公開）

「インターネット・ミニ解説」再生リスト

■ IIJ 公式Twitterアカウント@IIJ_ITS

なお、各媒体の記事公開やイベント開催情報は、IIJ公式Twitterアカウント@IIJ_ITSでお知らせしています。ご興味のある方はぜひフォローしてください。

@IIJ_ITS　https://twitter.com/IIJ_ITS

June 2022 Vol.55

IIJ
Internet Initiative Japan

株式会社インターネットイニシアティブ(IIJ)について

IIJは、1992年、インターネットの研究開発活動に関わっていた技術者が中心となり、日本でインターネットを本格的に普及させようという構想を持って設立されました。
現在は、国内最大級のインターネットバックボーンを運用し、インターネットの基盤を担うと共に、官公庁や金融機関をはじめとしたハイエンドのビジネスユーザに、インターネット接続やシステムインテグレーション、アウトソーシングサービスなど、高品質なシステム環境をトータルに提供しています。
また、サービス開発やインターネットバックボーンの運用を通して蓄積した知見を積極的に発信し、社会基盤としてのインターネットの発展に尽力しています。

本冊子の情報は2022年6月時点のものです。

IIJ-MKTG019-0055

株式会社インターネットイニシアティブ
〒102−0071 東京都千代田区富士見2-10-2　飯田橋グラン・ブルーム
E-mail : info@iij.ad.jp　URL : https://www.iij.ad.jp