HITACHI

Hitachi Systems Security Journal

VOL.13

株式会社 日立システムズ

Hitachi Systems Security Journal VOL.13

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

AI 時代の Web ブラウザーに潜む落とし穴
—— Edge Copilot のぜい弱性を解析

小勝 純 インタビュー

AI が組み込まれた最新の Web ブラウザーは、利便性を飛躍的に高める一方で、新たなリスクを内包している。Microsoft Edge に搭載された「Edge Copilot」は、ユーザー操作を支援する AI 機能だが、堅ろうに見えた内部構造の隙にぜい弱性が潜んでいた。CODE BLUE 2024 で講演した小勝氏は、AI 時代の Web ブラウザー設計に潜む落とし穴を解析し、XSS や権限委譲の不備が情報漏えいを招く実態を検証した。さらにインタビューでは、バグハンターの実情やゼロデイマーケットの現実にも迫る。

取材・文＝斉藤 健一／撮影＝卯月 梨沙

AI 搭載 Web ブラウザーのぜい弱性を明らかにした研究発表

斉藤（以下 S）：本日はよろしくお願いします。まず、講演内容について、読者に向けて、改めて、説明をお願いしたいと思います。まず、Edge Copilot とは、どのような仕組みで動作しているのですか？

小勝（以下 K）：Edge Copilot は、Edge ブラウザー内に組み込まれたチャットインターフェース型の AI アプリケーションで、アクティブなタブ上のコンテンツにアクセスし、閲覧中の Web ページを要約するなど、さまざまな操作ができます。内部的には「ディスカバーチャット」と呼ばれるブラウザー内部ページと Bing の iFrame が組み合わさって動作しており、Web UI と iFrame 間の通信は postMessage 経由で行われます。また、カメラやマイクへのアクセスがデフォルトでオンになっており、そこから特殊な API を呼び出せる構造になっています。

S Edge Copilot のセキュリティはどこに問題があるのでしょうか。

K 内部ページ自体は強力な CSP や Trusted Types を備えた堅ろうな SPA で、XSS を実質的に排除しています。しかし、統合されている Bing 側に複数のぜい弱性が残っており、そこを突かれると全体の安全性が崩れる点が本質です。

S セキュリティのぜい弱性を突くとどのようなことができますか。

K XSS を起点に新しいタブを開き、Copilot にプロンプトを送信して会話履歴を参照させ、それを外部へ送信するという一連のチェーンを作れる可能性があります。つまり、ユーザーの介入なしに任意のサイトのコンテンツが読み取られ、外部に漏えいするリスクが生じます。

S 具体的にはどのような攻撃が可能でしたか。

K 講演では 3 つの攻撃について解説しました。1 つめは Bing の XSS を使って新しいタブを開き、そのタブで「このページの内容を要約して」と Copilot に指示すると会話履歴が Bing と共有されます。実装上、ページ関連の質問でのみ履歴共有を抑制するべきところが不整合を起こし、関係ない質問でもページ内容が会話内に埋め込まれてしまうケースを確認しました。これを悪用すると、ユーザーの関与なしにページ内容を読み取られる可能性があります。

●小勝 純（Jun Kokatsu）
Web ブラウザーおよび Web セキュリティ・エンジニア／リサーチャー。フォロワーのいない思想的リーダー。現在、Google で Web および LLM セキュリティに従事している。以前は、Microsoft の Chromium Edge セキュリティチームの創設メンバーであった。
X（旧 Twitter）：@shhnjk

S 非常に興味深いですね。では、2 つめの攻撃についても教えていただけますか。

K Edge Copilot は、ページのタイトルが変わるたびに Web UI へ postMessage を送りますが、この機能に不備があり、HTML インジェクションを引き起こすことができました。結果として、フレームの permission delegation（カメラやマイクの権限委譲）を不正に利用する一連の連鎖が確認されています。つまり、XSS を起点にタブ操作を行い、プロンプトを送信して履歴を参照し、さらに外部へ送信する一連の流れに加えて、カメラやマイクの権限を不正に移譲できる可能性があったということです。

S 外部からカメラやマイクが使えてしまうというには驚きです。続いて 3 つめの攻撃についてもお聞かせください。

K 以前は任意のサイトから埋め込めた Edge の特

別エンドポイント（edgeservices…）に、URL ハッシュ変化でコマンドを受け取る仕組みがありました。そのコマンド経路を使ってプロンプト注入し、Copilot の「search memory（過去会話参照機能）」を誘導して過去会話を取り出させる手法を確認しました。取り出した情報を Markdown 内のリンクや画像に埋め込んで外部ロードさせる試みを行い、いくつかの構文で情報漏えいが可能な連鎖パターンを見つけました。画像経由はブロックされる例もありましたが、別の手段で隠蔽する工夫が機能するケースがありました）。

S どの部分に問題があるとお考えですか。

K Edge 側の設計は比較的堅ろうでしたが、Bing 側には依然として年に複数件の XSS などぜい弱性が存在しました。安全な部分と安全でない部分を統合すると全体の安全性は担保できないため、統合される両方のシステムが十分に安全であることが必須です。AI 特有のリスクも重要ですが、それ以前に基本的なぜい弱性（特に XSS）を徹底的に潰すことが最優先です。同時に、AI 経由の新しい漏えいパターンに対する設計上の緩和策（履歴共有の厳格化やプロンプト注入対策、権限委譲の厳格な制御など）を別途用意する必要があります。

調査の出発点とこれまでのキャリア

S 今回、Copilot を調べてみようと思われたきっかけは、どのようなところにあったのでしょうか？

K もともと Edge のセキュリティには以前から携わっており、その内部構造についても十分に理解していました。私がマイクロソフトを退職したのは 2022 年 6 月頃で、その少し後に AI ブームが始まりました。ちょうど同年 12 月頃に ChatGPT が登場し、「これは非常に興味深い展開になりそうだ」と感じたことが、今回の調査を始めるきっかけとなりました。

S なるほど。以前から Edge のセキュリティに携わっていたご経験が、今回の調査へとつながっているのですね。

K 自分が関わっていなかった分野において新たな機能が次々と登場する中で、特に注目すべき存在だったのが Copilot でした。ちょうど報奨金制度（バグバウンティ）も実施されていたことから、「少し調べてみよう」と思い立ったのです。もともと Edge のセキュリティに携わっていたこともあり、その内部構造について理解があったため、比較的取り組みやすい分野だと感じて調査を始めました。

小勝氏の講演動画やスライドなどは CODE BLUE 公式サイトのアーカイブページからアクセスできる
https://archive.codeblue.jp/2024/results/results/#result-2

S 今回の CODE BLUE が初めての発表の場となったのでしょうか。また、今回の調査には、どの程度の期間を費やされたのでしょうか。

K はい、今回の CODE BLUE が初めての発表となります。準備にはおよそ 1 年ほどを要しました。もともとプレゼンテーションの準備はあまり得意なほうではなく、文章を書くのも少し苦手なのです。

S それでも今回、発表しようと思ったのには何か理由があったのでしょうか。

K 今回の発表は、私が普段行っているバグハンティングの取り組みについて、その情報を共有する内容です。正直に言うと、「種明かし」をしてしまうと自分の収入が減ってしまう可能性もあります。それでも発表しようと思ったのは「日本のセキュリティコミュニティに少しでも貢献できたら」という気持ちがあったからです。とくに、これからの世代、若い人たちにはぜひ頑張ってほしいと思っています。

S ご自身の経験や知見を共有することで、次の世代の育成につなげようとするのは、素晴らしいお

考えですね。
K それに、CODE BLUE の講演に採択されると、チケット代やホテル代を負担してもらえるんです。そうしたサポートも、大きなモチベーションになっています。
S 現在はどちらにお住まいですか。
K 米国ワシントン州のシアトルです。
S マイクロソフトのお膝元ですね。
K そうです。もともと、マイクロソフトの米国本社にいましたから。Google に転職した現在でもシアトルに住んでいます。
S 元々、お住まいは米国だったのですか。
K いえ、父がパキスタン人、母が日本人という家庭に生まれ、中学生までは日本で生活していました。その後、英語を学ぶためにパキスタンの親族のもとへ渡り、そこからドバイの大学へ進学しました。就職は日本の通信キャリアのドバイ支店に現地採用という形でした。
S 海外で進学・就職されたのには何か理由があるのでしょうか。
K そもそも、日本の受験制度があまり好きではなかったというのが大きな理由です。日本の大学では多くの場合、入学試験に合格しなければなりませんが、海外の大学ではそれまでの活動や実績など、より多面的な評価が行われます。その点に魅力を感じました。
S 確かに。
K 受験のためだけに時間を費やさなければならない、という点が苦痛でした。私は「楽しくないことからは逃げる」というのを信条にしています。そのため、就職面接のような場もあまり得意ではありません。言葉で自分をアピールするよりも、実績で示すタイプなんです。

バグハンターのリアルライフ

S バグハンティングに興味を持ったきっかけは、どんなところにありましたか。
K もともと大学では IT 系の学科を専攻していました。2015 年頃だったと思いますが、ちょうどアノニマスが話題になり、さまざまな組織が DDoS 攻撃を受けたり、ハッキングされたり、PlayStation Network が乗っ取られたりといった出来事が相次いでいました。当時はそうしたニュースが頻繁に報じられており、「ハッキング」というものに強い関心を持つようになったのです。
S 当時のアノニマスは、IS（イスラム国）へのサイバー攻撃を宣言するなど、連日ニュースを賑わせていましたね。
K また、私生活の面でも大きな変化がありました。大学在学中に結婚し、その 1 年後に子どもが生まれたんです。ドバイでは、幼稚園から大学まで基本的にすべて私立で、公立校にはアラブ人の現地の方しか通うことができません。そのため教育費が非常に高く、「家族を支えるためにしっかり稼がなければ」と強く意識するようになりました。もともとセキュリティには関心があったので、「それならバグハンティングをやってみよう」と思い立ち、2015 年頃から本格的に取り組み始めました。
S 日本のセキュリティ・コミュニティとのつながりが生まれたのも、この頃だったのでしょうか
K 日本語でバグハンティングの情報を探していた際に、日本のコミュニティの方々が発信している情報に触れる機会がありました。また、この時期にサイボウズ社のバグ報奨金プログラムにも参加し、2015 年には報奨金獲得ランキングで 1 位となりました。サイボウズ社ではバグハンターを集めた合宿などのイベントも開催しており、そうした活動を通じて日本のセキュリティコミュニティとのつながりが生まれていきました。
S ちょうど報奨金のお話が出ましたが、今回の Copilot のぜい弱性についても、マイクロソフトから報奨金が支払われたとうかがっています。もし差し支えなければ、その金額について教えていただけますか。
K 今回、先ほどお話ししたとおり、3 つのぜい弱性を報告しています。1 つ目のぜい弱性が 2 万ドル、2 つ目が 1000 ドル、そして 3 つ目が 6000 ドルで、合計 2 万 7000 ドルの報奨金を獲得しました。
S 年によって金額にばらつきはあると思いますが、年間ではどのくらいの報奨金を得ているのでしょうか。
K 年間 5 万ドルから 6 万ドルくらいです。
S バグハントにどれくらいの時間を費やしているのですか。
K 日中はもちろん会社の業務がありますし、子ど

もも 2 人いますので、バグハントに充てられる時間は夜の 10 時から 12 時くらいの間です。それも毎日というわけではなく、気分が乗ったときに取り組む、といったペースですね。

S おおよその見込みで構いませんが、もしフルタイムでバグハンティングに専念した場合、どの程度の報奨金を得られると思いますか。

K おそらく、15 万ドル程度は達成できるのではないかと思います。

S 会社を辞めて、バグハンティングだけで生計を立てることを考えたことはありますか。

K ありません。子どももいますし、安定した職に就いていることは大切だと考えています。それに、現在の収入のほうが、先ほどお話しした予想額よりも多いんです。

S さまざまな情報を率直にお話しいただき、ありがとうございます。ちなみに、お知り合いの中には、バグハンティングだけで生計を立てている方はいらっしゃいますか。

K X（旧 Twitter）などではそうした方を多く見かけますが、私の知り合いの中にはいません。そもそも優秀な人材の多くは、企業に雇用されるケースがほとんどです。住んでいる国や地域の物価が比較的安く、企業に勤めるよりもバグハンティングをフルタイムで行ったほうが収入を得やすい環境にいる方々なのだと思います。

知られざるゼロデイマーケットの世界

S バグハンティングに関しては、必ずしもベンダーへの報告に限らず、ブラックマーケットやエクスプロイト開発企業などによる売買が話題になることもあります。ただ、このあたりの実情は一般の人にはなかなか見えません。もしご自身やお知り合いの方で、そうした事情をご存じでしたら、教えていただけますか。

K 以前はマイクロソフトで Edge のセキュリティを担当していたため、社内外を問わず、さまざまな方面から多くのぜい弱性報告を受けていました。Edge は Web ブラウザーの中でも特に攻撃対象として注目されており、ゼロデイマーケットの中でも最大級の分野を占めています。

S 確かに。Windows は PC の OS では圧倒的なシェアを誇りますし、Edge は標準搭載されています。攻撃者にとっては格好のターゲットになるわけですね。

講演内容にとどまらず、バグハンターの実情やゼロデイマーケットの動向などについてもオープンに語る小勝氏

K 知り合いの中には、かつてブラックマーケットでゼロデイぜい弱性を売却したことがある人や、逆にエクスプロイト開発企業に在籍していた人もいます。ぜい弱性の扱い方については、人それぞれの倫理観に委ねられる部分が大きいと思います。

S ぜい弱性の扱い方そのものだけでなく、そうした経歴やバックグラウンドを公にするという発想自体が、日本人にはあまりない感覚かもしれませんね。

K この倫理観を突き詰めていくと、「あなたは銃を作りたいですか」「あなたはお金のために銃を作れますか」と問われることに近いのだと思います。その銃が良いことに使われるか、悪いことに使われるかは分かりません。。

S ぜい弱性そのものに善悪はありませんが、それを使う人間の心には善悪があります。簡単に割り切れる話ではありませんね。

K 私は日本人的な感覚なので、それが良いことだとは思えません。確かに銃を作る人は必要かもしれませんが、自分では作りたくないという気持ちです。ただ、先ほども言ったように、豊かでない国や地域の人であったり、セキュリティ企業へ就職する道がなかったりすると、そうした選択肢を

取らざるを得ない人もいるのだと思います。

S 世界は豊かな先進国だけで成り立っているわけではないということを、改めて実感させられますね。

K 例えば、ルーターなどバグ報奨金制度のない分野にもゼロデイマーケットは存在します。また、高額でぜい弱性を買い取ってくれるという話も耳にします。そうした状況の中で、「高く買ってもらえるならそこに売ろう」と考える人が出てきても不思議ではありません。

S 倫理感は生まれ育ってきた環境や経済状況によって変わってきますね。ゼロデイマーケットの仕組みなどについて、ご存じであれば、可能な範囲でお話いただけますか。

K ぜい弱性の買い取りを公に行っている企業もありますが、それとは別に、政府向けのコントラクター（請負業者）がぜい弱性を買い取るケースも存在します。現在では、いわゆる「ぜい弱性ビジネス」が確立しており、さまざまな業態の企業が参入しています。例えば、発見されたバグを買い取って社内のデベロッパーがエクスプロイト化する企業や、政府向けにフルパッケージとして販売する業者もあります。中には、「このぜい弱性が使用できなくなった場合、類似のぜい弱性を継続的に提供する」あるいは「特定のブラウザーやアプリ向けに、1年間有効なエクスプロイトを提供する」といった年間契約で運用する企業もあり、多様なプレイヤーとビジネスモデルが混在しているのが実情です。

S 以前、ニューヨーク・タイムズ紙の記者が書いたゼロデイぜい弱性の売買に関するルポ※を読んだことがあるのですが、まさに同じ世界を感じました。今回、その一端を直接伺えて、とても興味深かったです。最後に、今後の目標をお聞かせください。

K 現在の目標は、AI分野におけるセキュリティ強化です。私は業務用Webセキュリティチームに所属していますが、近年はAI関連の研究にも力を入れています。Googleではすでに Gmail や社内ツールなどに AI が統合され、従来の XSS やインジェクションといったぜい弱性はほとんど見られなくなりました。しかし今度は、AI自体が新たな脅威を生み出し始めています。特に Gmail のようなセンシティブな領域にも AI が入り込み、そのリスクは無視できません。現在は「AI エージェントセキュリティチーム」と「Web セキュリティチーム」の両方に所属しており、今後の焦点はチャット型ではなく、AI が自律的に動く「エージェント型」の世界に移りつつあります。

S はい。まさに「AI エージェント元年」とも呼ばれるほど急速に発展してきていますね。今後のセキュリティにも大きな影響を与えそうです。

K AI にどこまで任せるか、どのタイミングでユーザー確認を挟むかといった設計判断が非常に重要になっています。AI の動作は確率的で、「90%防げる」防御策があっても、残りの10%で大きな影響を及ぼす可能性があります。Google のように数十億人のユーザーを抱える場合、わずか1%の失敗でも甚大な被害につながります。だからこそ、確率的な防御ではなく、セキュリティ・エンジニアリングの観点から設計された、より決定的な防御策が求められています。現在は AI モデルそのものではなく、「AI を組み込んだアプリケーション」が直面する脅威を分析し、プロンプトインジェクションなどをアプリケーション側でどう防ぐかを研究しています。こうした取り組みを通じて、この分野に貢献できればと思っています。

S 本日はありがとうございました。

※ **『サイバー戦争 終末のシナリオ（上下巻）』ニコール・パーロース 著、江口泰子 飜訳、岡嶋裕史 翻訳／監修）**
https://www.hayakawa-online.co.jp/shop/g/g0005210154/

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 2025.11

文＝日立システムズ

COTSを利用する宇宙産業を例にみた現状のリスクと今後

【概要】：低コスト・短期間での開発を目的に、さまざまな分野で市販の既製品COTS（Commercial Off-the-Shelf）が採用されている。安全性が求められる宇宙産業の小型衛星でも利用が進む一方、サプライチェーンリスクが指摘されている。2025年10月、米国の研究チームがリスク検証フレームワーク「SpyChain」と実証結果を公開した。今後、宇宙産業やCOTS関連事業者には、透明性・完全性・追跡性を重視した製造・検証体制が求められる。

【内容】：COTSとは、品質や安全性が厳しく求められる航空宇宙産業などで、専用品ではなく市販の既製品を採用する考え方、またはその製品を指す。COTSの利用により、調達コスト削減や開発期間短縮が可能となる。近年、鉄道・自動車・産業制御など多様な分野でも導入が進み、設計・製造の効率化や競争力向上に寄与している。しかし、十分に検証されていないサードパーティ製COTSの利用はサプライチェーンリスクを生む恐れがあり、脅威も現実化しつつある。一方、宇宙分野の衛星に関するリスクや脅威は報告されていない。こうしたリスクの実在性を検証するため、米国ニューメキシコ大学の研究チームはNASAの衛星シミュレーターを用いたフレームワーク「SpyChain」を提案した。同チームは5つの攻撃シナリオを実証し、その1つではCOTSに埋め込まれた悪意あるモジュールが地上試験や打上げ時は休眠し、軌道上到達など特定条件で活動を開始する仕組みを示した。これにより、ネットワーク経由とは異なるサプライチェーン上の脅威が宇宙産業にも存在することが確認された。こうした脅威の背景には、小型衛星で使われる組み込みOSや独自OSにおけるCOTSへの権限管理不足がある。OSで適切に権限管理が行われていれば、悪意あるモジュールの影響を軽減できると考えられる。

JAXAは、小型衛星の低コスト化・短期開発・競争力向上を目的に、COTSの宇宙転用を積極的に進めている。例えば「革新的衛星技術実証」プログラムでは、民生カメラやマイコンボードを用いた実証が行われ、非宇宙技術の導入が検討された。2025年には、小型衛星向けCOTS部品の選定・評価方法を体系化した「宇宙転用可能部品ハンドブック」が改訂され、信頼性評価や試験プロセスの標準化が進んでいる。サプライチェーンリスクへの対応として、COTS部品転用時は部品の由来や製造情報の確認、信頼性試験が必須とされている。一方、SpaceX社も外部依存を抑えつつサプライヤーと戦略的関係を築き、監査や品質管理、データ分析でリスクを監視している。しかし、供給元管理は十分でなく、下請け工程での確認不足も指摘される。

現行の経済安全保障推進法では、宇宙産業は基幹インフラを提供する特定社会基盤事業に指定されていない。しかし政府の宇宙技術戦略では、衛星通信を災害時のレジリエンス確保や国家安全保障の重要要素と位置づけている。そのため、将来的に特定社会基盤事業へ指定され、サプライチェーンリスク対応が義務化される可能性がある。結果として、宇宙産業の事業者だけでなく、COTS部品や素材の供給元にも透明性・完全性・追跡性を重視した体制が求められると考えられる。

［情報源］ https://arxiv.org/abs/2510.06535
https://www.mext.go.jp/content/20200721-mxt_uchukai01-000009195_2.pdf
https://sma.jaxa.jp/TechDoc/Docs/JAXA-JERG-2-027.pdf
https://spacecrew.com/space-jobs/m2oexuvo-spacex-sr-sourcing-specialist-ee-components-starlink
https://www.eurasiantimes.com/china-scientists-have-a-new-obsession/
https://www8.cao.go.jp/space/gijutu/honbun_20250325.pdf
https://www.cao.go.jp/keizai_anzen_hosho/suishinhou/infra/doc/infra_jigyousya.pdf

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

2. プロセス・ファイル監視編

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。前回より「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・分析する方法を整理します。対象とするコマンドは **ss, lsof, ps, ip, crontab, history, date, who** であり、どのコマンドもインシデント対応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早期の状況把握をするとともに、痕跡の保存・再構築といった対応を行なう必要があります。本稿では、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. プロセス監視に役立つ ps コマンド

2.1 ps：実行中のプロセス確認

ps（process status）は、Linux や UNIX 系 OS において、現在動作中のプロセス情報を表示するコマンドです。システムの監視やトラブルシューティング、さらには不正なプロセスの検出など、インシデント調査でも重要な役割を果たします。

なお、**ps** は一度だけスナップショットとして現在の状態を表示するコマンドです。リアルタイムでの表示が必要な場合は、**top** や **htop** が使用されます。

2.1.1 基本的な操作

・現在の端末に紐づくプロセスの表示

```
# ps
```

・BSD 形式で全プロセスを表示

```
# ps aux
```

・名前でプロセスを検索

```
#  ps –ef | grep [ プロセス名 ]
```

・特定のプロセス ID の情報を表示

```
#  ps –p [PID]
```

```
root         754  0.0  0.5 257888 20224 ?        Ssl  Apr21   0:04 /usr/sbin/NetworkManager --no-daemon
root         779  0.0  0.2  16668  9344 ?        Ss   Apr21   0:00 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
root         800  0.0  0.0   6832  3712 ?        Ss   Apr21   0:00 /usr/sbin/crond -n
root         801  0.0  0.1  13900  6272 ?        Ss   Apr21   0:00 login -- root
root         848  0.0  0.2 164732  8832 ?        Ssl  Apr21   0:09 /usr/sbin/rsyslogd -n
root        4477  0.0  0.3  23748 13688 ?        Ss   Apr21   0:00 /usr/lib/systemd/systemd --user
root        4479  0.0  0.1 109088  7008 ?        S    Apr21   0:00 (sd-pam)
root        4486  0.0  0.1   5616  4224 tty1     Ss   Apr21   0:02 -bash
root        4519  0.0  0.0   2680  1280 tty1     S    Apr21   0:00 tail -f /tmp/testfile.txt
root        4659  0.0  0.0      0     0 ?        I<   Apr21   0:00 [kworker/R-tls-s]
root       15350  0.0  0.1  13900  6272 ?        Ss   Apr22   0:00 login -- root
root       15354  0.0  0.1   5616  3968 tty2     Ss+  Apr22   0:00 -bash
root       15644  0.0  0.0      0     0 ?        I    Apr22   0:01 [kworker/u10:0-events_unbound]
root       15676  0.0  0.0      0     0 ?        I    Apr22   0:00 [kworker/u9:2-events_unbound]
root       15926  0.0  0.0      0     0 ?        I    Apr22   0:00 [kworker/u10:2-events_unbound]
root       15960  0.0  0.0      0     0 ?        I    Apr22   0:00 [kworker/u9:1-events_unbound]
root       15998  0.1  0.0      0     0 ?        R    00:33   0:32 [kworker/1:2+events]
root       16080  0.0  0.0      0     0 ?        I    01:30   0:00 [kworker/u10:3]
root       16193  0.0  0.0      0     0 ?        I    04:58   0:00 [kworker/0:2-events]
root       16209  0.0  0.0      0     0 ?        I    05:09   0:00 [kworker/1:0-kdmflush/253:0]
root       16210  0.0  0.0      0     0 ?        I    05:09   0:00 [kworker/0:1-ata_sff]
```

ps aux の出力画面

表 1 ps コマンドの出力の見方

項目	説明
USER	プロセスを実行しているユーザー名
PID	プロセス ID（一意の識別子）
% CPU	プロセスが消費している CPU 使用率
% MEM	プロセスが消費しているメモリー使用率
VSZ	仮想メモリーサイズ（KB 単位）
RSS	実メモリーサイズ（Redient Set Size、KB 単位）
TTY	プロセスが属している端末（例：pts/0、? は端末なし）
STAT	R：実行中、S：待機中など
START	プロセスの開始時刻（日付または時刻）
TIME	プロセスが使用した CPU 時間の累計
COMMAND	実行されるコマンド（引数付き）

2.1.2 ハンズオン

デフォルトの状態で **ps** コマンドを実施することでも機能を確認できますが、今回は自身でプロセスを作成し確認します。
指定した時間だけ処理を停止する **sleep** コマンドを利用します。
下記のようにコマンドを実行します。

```
# sleep 300 &
```

```
[root@localhost ~]# sleep 300 &
[1] 4517
```

実行後、下記コマンドを入力することでプロセスを確認することができます。

```
# ps -ef | grep sleep
```

```
[root@localhost ~]# ps -ef | grep sleep
root        4517    4486  0 13:42 tty1     00:00:00 sleep 300
root        4522    4486  0 13:43 tty1     00:00:00 grep --color=auto sleep
```

```
# ps
```

```
[root@localhost ~]# ps
    PID TTY          TIME CMD
   4486 tty1     00:00:00 bash
   4517 tty1     00:00:00 sleep
   4518 tty1     00:00:00 ps
```

コマンド実行後、下記のコマンドによりプロセスを終了してください。
%1 はプロセスの 1 番目を表す **jobs** と入力し、どこにプロセスがあるか確認した後、削除することを推奨します。

```
# kill %1
```

```
[root@localhost ~]# kill %1
[root@localhost ~]# ps
    PID TTY          TIME CMD
   4486 tty1     00:00:00 bash
   4524 tty1     00:00:00 ps
[1]+  Terminated              sleep 300
```

2.1.3 ps を利用したインシデント調査

ps コマンドのインシデント調査活用として、不審なプロセスを確認・記録・保全を実施します。
まず、ログ保存用ディレクトリを作成します。ファイル名は任意で問題ありません。実際のインシデント調査の場合は、調査した時刻を含めることを推奨します。
ps コマンドを入力し、全プロセスの詳細を取得しログ化します。

```
# mkdir -p ~/incident_logs/ps_20250508
# ps aux > ~/incident_logs/ps_20250508//ps_aux.txt
```

攻撃者が仕込んだ不審なプロセスを模倣し、調査手順を実践します。
疑似的な不審プロセスを手動で生成し、ps コマンドを使って検出・記録・保全を行います。
これにより、実際のインシデント対応を想定した調査スキルの習得を目指します。
まず、模倣した不審なプロセスを作成します。

```
# ps aux | grep bash
# ps -eo pid,user,command --sort=start_time | grep bash
```

```
[root@localhost ~]# ps aux | grep bash
root        4486  0.0  0.1  15332  7296 tty1     Ss   Apr21   0:02 -bash
root       15354  0.0  0.1   5616  3968 tty2     Ss+  Apr22   0:00 -bash
root       16224  0.0  0.0   4320  3200 tty1     S    05:34   0:00 bash -c while true; do sleep 10; done
root       16231  0.0  0.0   3332  1664 tty1     S+   05:35   0:00 grep --color=auto bash
[root@localhost ~]# ps -eo pid,user,command --sort=start_time | grep bash
   4486 root     -bash
  15354 root     -bash
  16224 root     bash -c while true; do sleep 10; done
  16238 root     grep --color=auto bash
```

command：実行内容を調査 **user**：誰が動かしているかを確認

確認した際、PID を特定し実行ユーザー、コマンドパスなどをチェックします。

```
# ps -p 16224 -o pid,ppid,user,stime,etime,cmd
```

```
[root@localhost ~]# ps -p 16224 -o pid,ppid,user,stime,etime,cmd
    PID    PPID USER     STIME     ELAPSED CMD
  16224    4486 root     05:34       03:52 bash -c while true; do sleep 10; done
```

PID に不審なプロセスを作成した際に表示された数字を入れてください。

最後に、調査ログとして最初に作成したディレクトリに保存することで証拠保全が完了します。
最初に作成したディレクトリに保存しました。

```
# ps aux > ~/incident_logs/ps_20250508//ps_aux_after.txt
```

3. ファイル監視に役立つ lsof コマンド

3.1 lsof：プロセスが開いているファイルやソケットなどを一覧表示

Linux システムでは、多くのプロセスがファイルやネットワークリソースを使用して動作しています。
lsof（List Open Files）は、現在システム上で開かれているファイルを一覧表示するコマンドであり、プロセスがどのようなファイルやソケットを使用しているかを把握できます。日常の運用監視やトラブルシューティングにとどまらず、インシデント調査においても有効なツールです。
タイムスタンプ付きのファイル名で保存しておくことで、後の時系列分析や証拠提出にも活用できます。

3.1.1 基本的な操作

・開いているすべてのファイルを表示

```
# lsof
```

・特定のユーザーが開いているファイルを表示

```
# lsof -u root
```

・特定のファイルにアクセスしているプロセス

```
# lsof /path/to/file
```

lsof は、現在システム上で開かれているすべてのファイルが一覧表示されます。ただし、出力が非常に多くなるため、利用時には特定の条件を付けて絞り込みながら利用することが多くあります。

```
lsof      16770            root  mem       REG              253,0  2544200   25921793 /usr/lib64/libc.so.6
lsof      16770            root  mem       REG              253,0    23992   25922043 /usr/lib64/libkeyutils.so.1.10
lsof      16770            root  mem       REG              253,0    66312   26076239 /usr/lib64/libkrb5support.so.0.1
lsof      16770            root  mem       REG              253,0    23872   25921973 /usr/lib64/libcom_err.so.2.1
lsof      16770            root  mem       REG              253,0    99120   26076231 /usr/lib64/libk5crypto.so.3.1
lsof      16770            root  mem       REG              253,0   636848   25922181 /usr/lib64/libpcre2-8.so.0.11.0
lsof      16770            root  mem       REG              253,0   196224   26425738 /usr/lib64/libtirpc.so.3.0.0
lsof      16770            root  mem       REG              253,0   175744   25922185 /usr/lib64/libselinux.so.1
lsof      16770            root  mem       REG              253,0   897544   25921725 /usr/lib64/ld-linux-x86-64.so.2
lsof      16770            root    0u      CHR                4,1      0t0         20 /dev/tty1
lsof      16770            root    1u      CHR                4,1      0t0         20 /dev/tty1
lsof      16770            root    2u      CHR                4,1      0t0         20 /dev/tty1
lsof      16770            root    3r      DIR               0,19        0          1 /proc
lsof      16770            root    4r      DIR               0,19        7      50174 /proc/16770/fd
lsof      16770            root    5w     FIFO               0,13      0t0      51203 pipe
lsof      16770            root    6r     FIFO               0,13      0t0      51204 pipe
lsof      16771            root  cwd       DIR              253,0     4096   16908421 /root
lsof      16771            root  rtd       DIR              253,0      235        128 /
lsof      16771            root  txt       REG              253,0   179232   17376958 /usr/bin/lsof
lsof      16771            root  mem       REG              253,0  5436968   25922538 /usr/lib64/libcrypto.so.3.2.2
lsof      16771            root  mem       REG              253,0   102672   25921920 /usr/lib64/libz.so.1.2.11
lsof      16771            root  mem       REG              253,0    68480   25921804 /usr/lib64/libresolv.so.2
lsof      16771            root  mem       REG              253,0   906400   26076237 /usr/lib64/libkrb5.so.3.3
lsof      16771            root  mem       REG              253,0   358584   26076227 /usr/lib64/libgssapi_krb5.so.2.2
lsof      16771            root  mem       REG              253,0  2544200   25921793 /usr/lib64/libc.so.6
```

lsof の出力画面

・特定のポート（TCP 80 番）を使っているプロセスを調べる

```
# lsof -i:80
```

・特定のプロセス ID に対して使う場合

```
# lsof -p <PID>
```

表 2 lsof コマンドの出力の見方

項目	説明
COMMAND	使用しているコマンド名（プロセス名）
PID	プロセス ID（一意の識別子）
USER	実行ユーザー
FD	ファイルディスクリプター（0=stdin, 1=stdout, 2=stderr など）
TYPE	ファイルタイプ （REG= 通常ファイル、CHR= キャラクターデバイス、IPv4= ネットワーク など）
RSS	実メモリーサイズ（Redient Set Size、KB 単位）
TTY	プロセスが属している端末（例：pts/0、? は端末なし）
DEVICE	デバイス番号
SIZE/OFF	ファイルサイズまたはオフセット
NODE	i-node 番号またはソケットの識別子
NAME	ファイル名、ポート、ソケット接続先 など

3.1.2 lsof を利用した調査方法

Linux では、ファイルが削除されていても、プロセスがそのファイルを開いたままであれば、そのファイルはディスクから完全に消えません。これにより、攻撃者が証拠隠滅のためにファイルを削除しても、プロセスが使用中であれば痕跡を確認できる可能性があります。また、特定のログファイルに関連するプロセスの特定や、不審なポートで通信しているプロセスの確認が可能です。

ハンズオンでは、ファイルを削除した場合でも、該当プロセスが開いていれば情報を表示できることを確認します。

ハンズオンを実施するにあたり、**lsof** は環境によって標準でインストールされていません。今回検証で利用している CentOS9 Stream には装備されていないため、ダウンロードが必要です。

ご自身の環境に存在するか下記のコマンドを実行し、ご確認ください。

```
# which lsof
```

lsof のダウンロードは以下のコマンドを実行してください。

```
# sudo yum install lsof –y
# sudo yum install lsof –y
```

まず、テスト用ファイルを用意し、ファイルを開いたまま保持するプロセスを作成します。

今回、執筆者は "tail" コマンドを利用しました。

tail -f は指定したファイルに新しい文章が追加されるたびに、その部分を表示するコマンドで、ログファイルのリアルタイム監視に広く利用されています。

```
# echo "secret data" > /tmp/testfile.txt
# tail –f /tmp/testfile.txt &
# secret data
```

secret data はお好きな文字に置き換えてください。

```
[root@localhost ~]# echo "secret data" > /tmp/testfile.txt
[root@localhost ~]# tail -f /tmp/testfile.txt &
[1] 14410
[root@localhost ~]# secret data
```

次に、作成したテストファイルの削除を行います。

```
# rm /tmp/testfile.txt
```

削除する際、2 行目のように削除してよいか尋ねられます。
y（=yes）と入力し、Enter を押してください。

```
[root@localhost ~]# rm /tmp/testfile.txt
rm: remove regular file '/tmp/testfile.txt'?
```

削除後、**lsof -c tail** や **lsof | grep deleted** で表示を行います。（deleted）の表記がありますが、プロセスはまだ中身を確認できます。

```
# lsof – c tail
```

```
[root@localhost ~]# lsof -c tail
COMMAND   PID USER   FD      TYPE DEVICE SIZE/OFF     NODE NAME
tail    14410 root  cwd       DIR  253,0      126 16908421 /root
tail    14410 root  rtd       DIR  253,0      235      128 /
tail    14410 root  txt       REG  253,0    69712 17263692 /usr/bin/tail
tail    14410 root  mem       REG  253,0  2544200 25921793 /usr/lib64/libc.so.6
tail    14410 root  mem       REG  253,0   897544 25921725 /usr/lib64/ld-linux-x86-64.so.2
tail    14410 root    0u      CHR    4,1      0t0       20 /dev/tty1
tail    14410 root    1u      CHR    4,1      0t0       20 /dev/tty1
tail    14410 root    2u      CHR    4,1      0t0       20 /dev/tty1
tail    14410 root    3r      REG  253,0       12  8757083 /tmp/testfile.txt (deleted)
tail    14410 root    4r  a_inode   0,14        0       41 inotify
[root@localhost ~]# lsof | grep deleted
dbus-brok   718                 dbus   12u      REG              0,1  2097152       1025 /memfd:dbus-broker-log (deleted)
firewalld   722                 root    9u      REG              0,1     4096          2 /memfd:libffi (deleted)
firewalld   722 832 gmain       root    9u      REG              0,1     4096          2 /memfd:libffi (deleted)
tail      14410                 root    3r      REG            253,0       12    8757083 /tmp/testfile.txt (deleted)
```

このように、攻撃者が証拠隠滅を試みた場合でも **lsof** を利用することで削除されたファイルを検出することが可能です。さらに、プロセス ID などから追跡を行うことで証拠性の高い情報や攻撃の手口を判断する材料になります。

4. タイムスタンプを確認する date コマンド

4.1 **date**：時刻の確認と設定

date コマンドは、Linux システムにおいて「現在のシステム日時の確認」「フォーマット付きの日時出力」「時刻の設定・変更」などに用いられる基本コマンドです。
システム管理におけるログの解析や、ファイルのタイムスタンプ確認、インシデント調査における証拠の正確な時間確認に利用できます。

4.1.1 基本的な操作

・現在の日時確認

```
# date
```

・ログ記録用にフォーマット

```
# date “%Y%m%d_%H%M%S”
```

・UTC での時刻表示 , ログとの差異確認

```
# date –u
```

4.1.2 証拠保全のための時刻取得

このハンズオンでは、**date** コマンドを活用して、調査用ログの記録開始時間を明記したログディレクトリを作成し、他のコマンドと組み合わせて時系列の証拠管理を実施します。

1. 時刻付きのディレクトリ作成

調査開始時点でログや証拠の保存先を明確に確保し、「いつの時点の調査か」を時刻ベースでラベリングするのに利用できます。

```
# LOGDIR=~/investigation_$(date "+%Y%m%d_%H%M%S"")

# echo "$L"
```

```
[root@localhost ~]# LOGDIR=~/investigation_$(date "+%Y%m%d_%H%M%S")
[root@localhost ~]# mkdir "$LOGDIR"
[root@localhost ~]# echo "$LOGDIR"
/root/investigation_20250423_073543
```

2. 時刻付きのディレクトリ作成

タイムスタンプを含むログやファイルと整合性を確認に利用できます。

```
# date > "$LOGDIR /start_time.txt"
```

3. 他のコマンド結果と一緒に保存

取得したプロセス一覧が「どの時点の状態か」を明示します。

攻撃プロセスが何時に実行されていたのか、他のログと比較ができます。

今回は ”ps” コマンドと同時に利用したケースを示します。

```
# ps aux  >"$LOGDIR/ps_snapshot.txt"
           "$LOGDIR/ps_snapshot.txt"
```

4. UTC 表示で記録

ログが UTC で記録されているシステムや異なるタイムゾーンの環境での調査時に利用します。

また、自国の基準を明示することで、解析ミスも防ぐことができます。

```
# date–u > "$LOGDIR /start_time_UTC.txt"
```

このハンズオンは、証拠保全・調査記録・時系列整合性の確保という観点で、実際のインシデント対応時に「何時に何があったかを裏付ける」ために極めて重要です。

4. おわりに

今回はここまでとなります。

本稿では、Linux における基本的な調査・分析コマンドである **ps,lsof,date** について、それぞれの機能や使用方法、そして実際のハンズオンを通じた活用例を取り上げました。これらのコマンドは一見すると単純な出力を行うだけに見えますが、インシデント対調査の場面においては非常に強力な調査手段となります。

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