IIR

Internet
Infrastructure
Review

Dec.2019
Vol.45

定期観測レポート

IIJインフラから見る インターネットの傾向～2019年

フォーカス・リサーチ

Linuxのフォレンジック向け メモリイメージ取得

IIJ
Internet Initiative Japan

Internet Infrastructure Review

December 2019 Vol.45

エグゼクティブサマリ

2019年最後の「IIR」をお届けします。今年の日本は天災、特に雨に関わる災害が多く発生しました。8月の九州北部豪雨、9月に関東を中心に広範な被害をもたらした台風15号及び19号は、記憶に新しいところです。地球温暖化という言葉を聞くと、気温の上昇がクローズアップされがちですが、大気中の温室効果ガスが増えることにより、大気の状態が全体的に変化し、雨の降り方も徐々に変わっていくのだそうです。

情報通信の業界においても、機器の省電力化、データセンターの省エネルギー化などの対策が行われています。そうしたなか、理化学研究所のスーパーコンピュータ「富岳」の試作機が、消費電力性能を示すランキング「Green 500」で世界一を実証したとのニュースがありました。

情報通信機器の省エネルギー化、情報通信を利用した社会の省エネルギー化など、持続可能な社会を実現するうえで、情報通信が果たせる役割は多く、私たちも技術開発に取り組んでいきたいと考えています。

「IIR」では、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、IIJインフラから見るインターネットの傾向の2019年版です。インターネット上のIPv4経路数の状況、利用者に提供しているフルリゾルバから得られるDNSのクエリの分析、IIJバックボーンにおけるIPv6の利用状況、自然災害発生時のモバイル網やフレッツ網のトラフィック状況、IIJのバックボーンの歴史などを紹介しています。

IPv4においては、経路広報されているユニークなアドレス数の減少が初めて観測され、今後の注視が必要です。IPv6は、その利用が着実に進んでいることが、トラフィックの量や多くの事業者による利用の増加を通して確認できました。また、台風19号が通過した10月12日前後のトラフィックを分析したところ、明らかに平時とは違った傾向が出ていることが分かりました。

2章の「フォーカス・リサーチ」では、私たちがインシデントレスポンスやフォレンジックでメモリ解析を行う場合に利用しているVolatilityにおいて、Linuxのメモリイメージを取得するツールを解説しています。Linuxのメモリイメージ取得ツールのうち、LiMEとcrashを紹介するとともに、ディスクフォレンジックに極力影響を与えないメモリイメージ取得方法についても解説しています。

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続けています。今後も企業活動のインフラとして最大限に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 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。

IIJインフラから見るインターネットの傾向 〜2019年

IIJではインターネットサービスを提供するために、国内でも有数規模のネットワーク・サーバインフラを運用しています。ここでは、IIJのインフラ運用を通じて得られた情報を元に、現在のインターネットがどのような傾向を持っているのかを検討し、報告します。

取り上げるテーマは、ネットワークの経路情報、DNS問い合わせ情報、IPv6利用状況の他、モバイル・フレッツ接続サービスにおける自然災害の影響についても述べます。また、IIJのトラフィックの大部分を支えるバックボーンネットワークの歴史についても併せて紹介します。

Theme 01

BGP・経路数

最初にIIJ網から他組織に広報している「IPv4 フルルート」の情報を確認します（表-1）。今回は「IPv4 フルルート」に含まれるunique IPv4アドレス数の情報も追加しました（表-2）。なおこの1年の間にAPNIC（及びJPNIC）ではIPv4アドレス割り振りサイズの上限が/23（512アドレス）に縮小されています。

経路総数は昨年と比べて若干少ない増加となりましたが76万を超えました。/22-/24の3プレフィクスが経路総数に占める割合は80.1%になっています。一方でunique IPv4アドレス数は総数の1%未満ではありますが過去9年間で初めての減少となりました。RPKIによる不正経路排除の効果など一時的なものなのか、はたまた『IPv4インターネットの収縮』の始まりなのか、今後はこちらも注視したいと思います。

表-1「IPv4 フルルート」に含まれるプレフィクス長ごとの経路数の推移

年月	/8	/9	/10	/11	/12	/13	/14	/15	/16	/17	/18	/19	/20	/21	/22	/23	/24	total
2010年9月	20	10	25	67	198	409	718	1308	11225	5389	9225	18532	23267	23380	30451	29811	170701	324736
2011年9月	19	12	27	81	233	457	794	1407	11909	5907	9885	19515	26476	26588	35515	34061	190276	363162
2012年9月	19	14	29	84	236	471	838	1526	12334	6349	10710	20927	30049	31793	42007	39517	219343	416246
2013年9月	16	11	30	93	250	480	903	1613	12748	6652	10971	22588	32202	34900	48915	42440	244822	459634
2014年9月	16	12	30	90	261	500	983	1702	13009	7013	11659	24527	35175	37560	54065	47372	268660	502634
2015年9月	18	13	36	96	261	500	999	1731	12863	7190	12317	25485	35904	38572	60900	52904	301381	551170
2016年9月	16	13	36	101	267	515	1050	1767	13106	7782	12917	25229	38459	40066	67270	58965	335884	603443
2017年9月	15	13	36	104	284	552	1047	1861	13391	7619	13385	24672	38704	41630	78779	64549	367474	654115
2018年9月	14	11	36	99	292	567	1094	1891	13325	7906	13771	25307	39408	45578	88476	72030	400488	710293
2019年9月	10	11	37	98	288	573	1142	1914	13243	7999	13730	25531	40128	47248	95983	77581	438926	764442

表-2「IPv4 フルルート」に含まれるunique IPv4アドレス総数の推移

年月	IPv4 アドレス数
2010年9月	2,277,265,152
2011年9月	2,470,856,448
2012年9月	2,588,775,936
2013年9月	2,638,256,384
2014年9月	2,705,751,040
2015年9月	2,791,345,920
2016年9月	2,824,538,880
2017年9月	2,852,547,328
2018年9月	2,855,087,616
2019年9月	2,834,175,488

次に「IPv6 フルルート」の情報を確認します（表-3）。経路総数は昨年よりも大きな伸びを示し7万を超えました。ただし分割広報された経路が全体の過半数を占める状況は変わっておらず、割り振り/割り当てサイズとしてはあまり見ることのないプレフィクス長"/30-/31"、"/41-/43"、"/45-/47"が昨年からの増加率トップ3となっています。

最後に「IPv4/IPv6 フルルート」広報元AS（Origin AS）数を確認します（表-4）。16-bit AS番号Origin ASの減少数、32-bit onlyAS番号Origin ASの増加数、共に過去9年間で最大となりました。またIPv6経路を広報するAS ("IPv6-enabled")が全体の1/4を初めて超えました。RIPE NCCのIPv4アドレス在庫が2020年を待たずに完全に枯渇するとの予測も出ており、次回どうなるのか楽しみです。

Theme 02

DNSクエリ解析

IIJでは利用者がDNSの名前解決を利用できるようフルリゾルバを提供しています。この項目では名前解決の情況を解説し、IIJで2019年10月25日に行ったフルリゾルバの1日分の観測データから、主にコンシューマサービス向けに提供しているサーバのデータに基づいて分析と考察を行います。

フルリゾルバはrootと呼ばれる最上位のゾーン情報を提供する権威ネームサーバのIPアドレスを手がかりとして、そこから得られる情報に基づき権威ネームサーバをたどって必要なレコードを探します。フルリゾルバで毎回反復問い合わせを行っていると負荷や遅延が問題となるため、得られた情報はしばら

表-3「IPv6 フルルート」に含まれるプレフィクス長ごとの経路数の推移

年月	/16-/28	/29	/30-/31	/32	/33-/39	/40	/41-/43	/44	/45-/47	/48	total
2010年9月	38	3	10	2023	33	2	9	4	17	436	2575
2011年9月	68	13	22	3530	406	248	45	87	95	2356	6870
2012年9月	102	45	34	4448	757	445	103	246	168	3706	10054
2013年9月	117	256	92	5249	1067	660	119	474	266	5442	13742
2014年9月	134	481	133	6025	1447	825	248	709	592	7949	18543
2015年9月	142	771	168	6846	1808	1150	386	990	648	10570	23479
2016年9月	153	1294	216	8110	3092	1445	371	1492	1006	14291	31470
2017年9月	158	1757	256	9089	3588	2117	580	1999	1983	18347	39874
2018年9月	168	2279	328	10897	4828	2940	906	4015	2270	24616	53247
2019年9月	192	2671	606	12664	6914	3870	1566	4590	4165	34224	71462

表-4「IPv4/IPv6 フルルート」の広報元AS数の推移

AS番号	16-bit(1〜64495)					32-bit only(131072〜4199999999)				
広報経路	IPv4+IPv6	IPv4のみ	IPv6のみ	total	(IPv6 -enabled)	IPv4+IPv6	IPv4のみ	IPv6のみ	total	(IPv6 -enabled)
2010年9月	2083	32399	67	34549	(6.2%)	17	478	3	498	(4.0%)
2011年9月	4258	32756	115	37129	(11.8%)	90	1278	13	1381	(7.5%)
2012年9月	5467	33434	125	39026	(14.3%)	264	2565	17	2846	(9.9%)
2013年9月	6579	34108	131	40818	(16.4%)	496	3390	28	3914	(13.4%)
2014年9月	7405	34555	128	42088	(17.9%)	868	4749	55	5672	(16.3%)
2015年9月	8228	34544	137	42909	(19.5%)	1424	6801	78	8303	(18.1%)
2016年9月	9116	33555	158	42829	(21.7%)	2406	9391	146	11943	(21.4%)
2017年9月	9603	32731	181	42515	(23.0%)	3214	12379	207	15800	(21.7%)
2018年9月	10199	31960	176	42335	(24.5%)	4379	14874	308	19561	(24.0%)
2019年9月	10642	31164	206	42012	(25.8%)	5790	17409	432	23631	(26.3%)

くキャッシュしておいて再び同じ問い合わせを受けた場合にはそのキャッシュから応答しています。最近はこの他にもブロードバンドルータやファイアウォールなど、通信経路上の機器にもDNS関連の機能が実装されており、DNS問い合わせの中継や制御ポリシーの適用に関わっている場合があります。

ISPは接続種別に応じてPPPやDHCP、RA、PCOなどの通知手段を利用してフルリゾルバのIPアドレスを利用者に伝え、利用者が名前解決用のフルリゾルバを端末で自動設定できるようにしています。ISPは複数のフルリゾルバを利用者に伝えられる他、利用者は自身でOSやWebブラウザなどの設定を変更して利用するフルリゾルバを指定、追加することもできます。端末に複数のフルリゾルバが設定されている場合、どれを利用するかは端末の実装やアプリケーションに依存するため、フルリゾルバ側では利用者が総量としてどの程度の問い合わせを行っているか分かりません。このため、フルリゾルバでは問い合わせ動向を注視しながら、常に処理能力に余裕を持たせて運用する必要があります。

IIJが提供するフルリゾルバの観測データを見てみると、利用者の利用傾向を示すように時間帯によって問い合わせ量が変動し、朝4時半頃に問い合わせ元のIPアドレス当たり最小の0.05query/sec、昼12時半頃にピークを迎えて0.23query/sec程度になっています。この値は昨年とほぼ同様ですが、ピーク値が+0.01ポイントと若干上昇しています。問い合わせ傾向を通信に使われたIPv4とIPv6のIPプロトコル別に見てみると、深夜帯は大きな違いはなくほぼ同じ傾向を示している一方、人の活動する日中や特に20時以降では顕著にIPv6でIPアドレス当たりの問い合わせが増える傾向が見えています。家庭でIPv6が利用できる環境が整備されてきていることを示唆していると考えています。また全体の問い合わせ数で見ると、IPv6による問い合わせが、問い合わせ元IP数、実際の問い合わせ数共にIPv4よりも多くなっています。IPv6による問い合わせ数は増加傾向にあり、昨年は全体の55%、今年は4ポイント以上増えて、全体の約60%がIPv6による問い合わせとなっています。

近年の特徴的な傾向として、朝方の毎正時など切りの良い時刻に一時的に問い合わせが増加しています。問い合わせ元数も同時に増えているため、利用者の端末でタスクをスケジュールしたり、目覚まし機能などで端末が起動することに伴う機械的なアクセスが増えたりといったことが原因だと推測できます。昨年は毎正時の14秒前に問い合わせが増加していましたが、今年はそれに加えて毎正時の10秒前にも問い合わせが増加しています。毎正時では増加後、緩やかに問い合わせ量が減っていくのに比べて、毎正時の14秒前と10秒前の増加ではすぐにそれまでの問い合わせ量程度に戻っています。つまり多くの端末が綺麗に同期して問い合わせを行っていることから、何かすぐに完了する軽量なタスクが実行されていると考えられます。

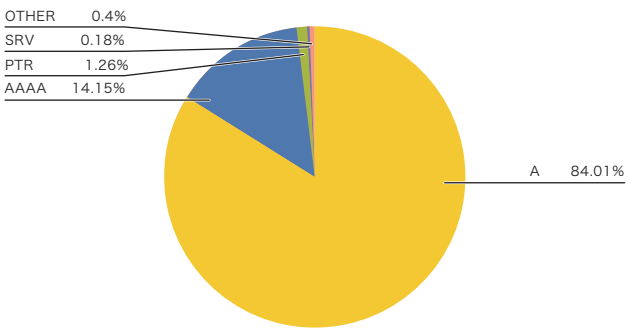

図-1 クライアントからのIPv4による問い合わせ

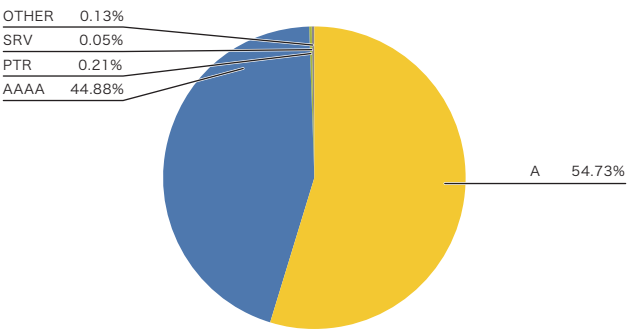

図-2 クライアントからのIPv6による問い合わせ

例えば接続確認や時刻同期など基本的なタスクを本格的なスリープ解除前に終わらせるような機構があり、これに利用されている問い合わせが影響していると予想しています。

問い合わせレコードタイプに注目すると、ホスト名に対応するIPv4アドレスを問い合わせるAレコードとIPv6アドレスを問い合わせるAAAAレコードがほとんどを占めています。AとAAAAの問い合わせ傾向は通信に利用されるIPプロトコルで違いが見られ、IPv6での問い合わせではより多くのAAAAレコード問い合わせが見られます。IPv4での問い合わせでは、全体の84%程度がAレコード問い合わせ、14%程度がAAAAレコード問い合わせです(図-1)。一方IPv6での問い合わせでは、全体の54%程度がAレコード問い合わせ、44%程度がAAAAレコード問い合わせと、AAAAレコード問い合わせの比率が高まっています(図-2)。昨年と比べると、IPv6はほぼ同様の傾向を示している一方、IPv4では3ポイント程度Aレコードの問い合わせが減り、AAAAレコードの問い合わせが3ポイント程度増加しています。

Theme 03

IPv6

ここではIIJバックボーンのIPv6トラフィックの流量、流入元、主なプロトコルについて報告します。また、モバイルにおけるIPv6に関する新たな切り口として、端末OS(Apple iOS/Android)の違いによるIPv6接続状況についても解説します。

■ トラフィック

前回同様、IIJのコアPOP(東京・大阪・名古屋)のバックボーンルータで計測した、IPv4トラフィックとIPv6トラフィックを図-3に示します。期間は2018年10月1日から2019年9月30日までの1年間です。

IPv4のトラフィックはこの1年で約8%増加、IPv6のトラフィックは1年で約85%増加しました。全トラフィックに占めるIPv6の割合(図-4)は、約10%となり、昨年の約6%から4ポイント増加しています。また、IPv6トラフィックの割合のピークは

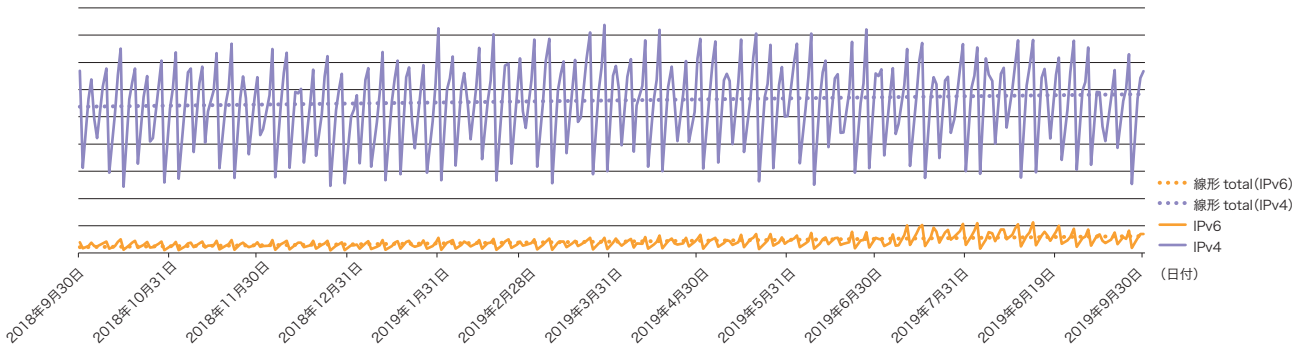

図-3 IIJのコアPOP(東京・大阪・名古屋)のバックボーンルータで計測した、IPv4トラフィックとIPv6トラフィック

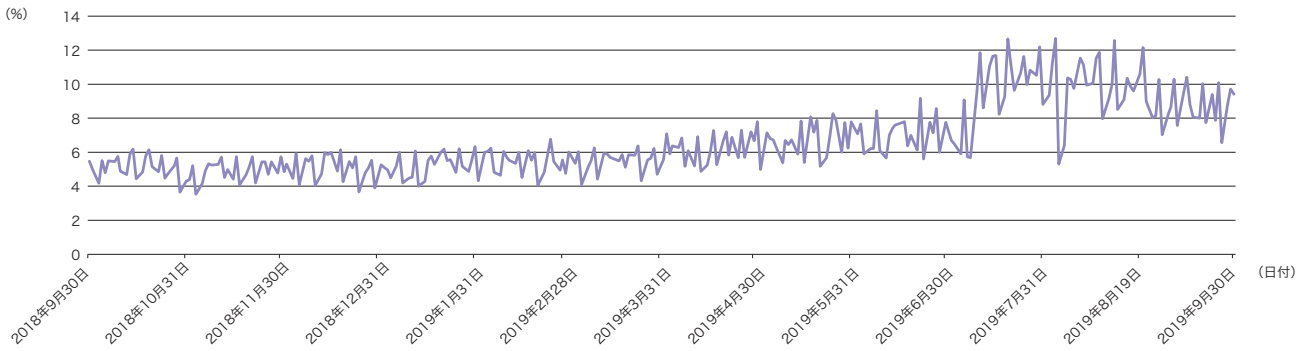

図-4 全トラフィックに占めるIPv6の割合

約12%を記録しており、IPv6トラフィックが占める割合は着実に増加しています。

図-5は同じ期間をログスケールで描画したものです。IPv4トラフィックの伸びが鈍化している中、IPv6トラフィックは着実に伸びてきているのが分かります。

■ 送信元組織(BGP AS)

次に、2018年10月から2019年9月までの1年間の、IPv6とIPv4の平均トラフィック送信元組織(BGP AS番号)の上位を図-6と図-7に示します。

やはり最上位はA社ですが、2位以下とのトラフィック量の差が更に縮小すると共に、占有率も前回比で6割程度に下がりました。これは、多くの事業者でIPv6の活用が進んでいると共に、2019年7月からIIJの関連会社で動画配信プラットフォームの提供を行うJOCDNのプラットフォームを利用している動画配信サービスでIPv6が有効化されたことで、IIJのIPv6トラフィックが増加したためと思われます。

■ 利用プロトコル

IPv6トラフィックのProtocol番号(Next-Header)と送信元ポート番号で解析したグラフを図-8に、IPv4トラフィックのProtocol番号と送信元ポート番号のグラフを図-9に示します(2019年9月30日からの1週間)。

IPv6において、前回3位だったTCP80(HTTP)が2位になり、UDP443(QUIC)が3位になりました。これは、IPv4と同じ並びになったということで、IPv6の使われ方がIPv4同様の傾向になってきた、あるいは一般的になったといっても良いでしょう。

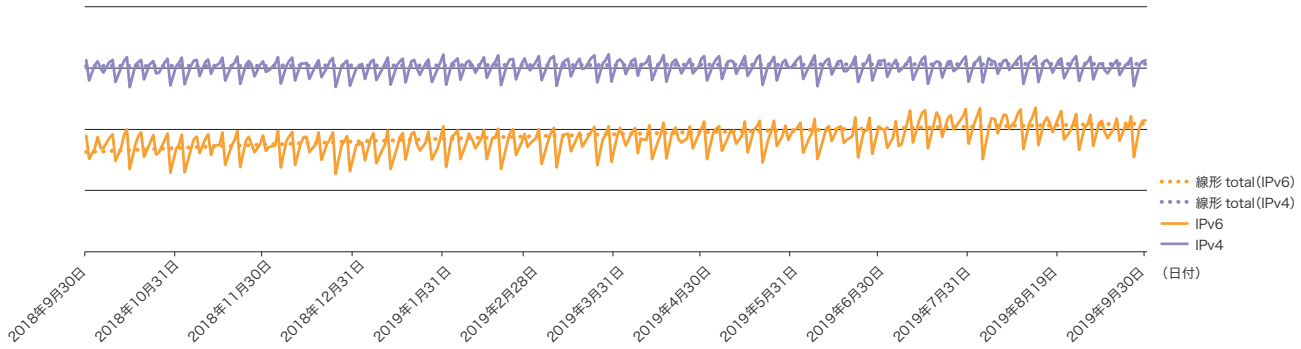

図-5 IIJのコアPOP(東京・大阪・名古屋)のバックボーンルータで計測した、IPv4トラフィックとIPv6トラフィック(ログスケール)

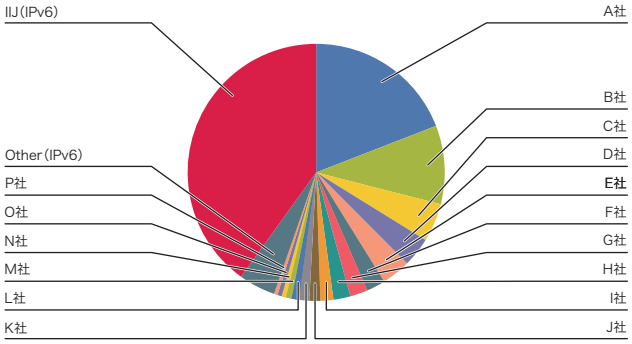

図-6 2018年10月から2019年9月までの
1年間の平均IPv6トラフィック送信元組織(BGPのAS番号)の上位

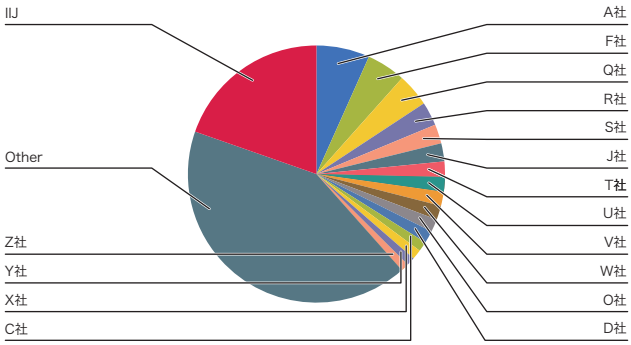

図-7 2018年10月から2019年9月までの
1年間の平均IPv4トラフィック送信元組織(BGPのAS番号)の上位

なお、前回はランキング外だったUDP4500が5位に入っています。IPv6ではNATは基本的に使われないはずですが、NATトラバーサルIPSec用として一般的に使われているUDP4500が上位に入っていることは興味深いところです。

■ モバイル端末OSの違いによるIPv6接続状況について

前回、「Apple iOSのバージョン11から、IPv6接続がデフォルトで有効化されたので、モバイルのIPv6トラフィックが増加した」ということを報告しました。

今回はモバイル端末のIMEI（International Mobile Equipment Identity）を解析し、OS種別（iOSかAndroidか）とIPv6接続しているかどうか（IPv6アドレスが割当られているかどうか）を分析してみました。

解析対象は個人向けモバイルサービス（IIJmioモバイルサービス）の約107万回線（2019年6月末時点の契約回線数）で、2019年10月の平日のとある日に接続中だったものです（MVNEとしての回線や法人契約回線は含みません）。

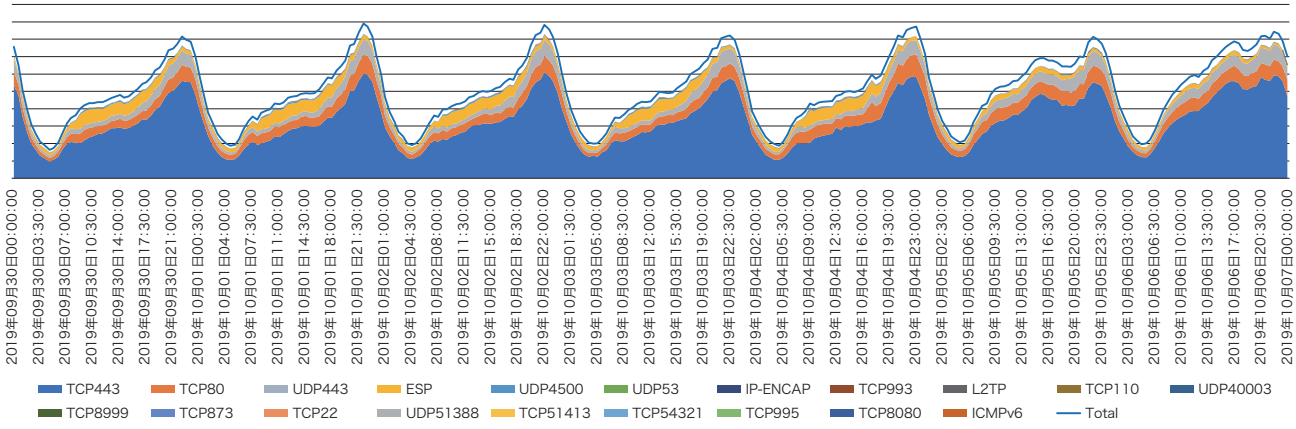

図-8 IPv6トラフィックのProtocol番号（Next-Header）と送信元ポート番号で解析したグラフ

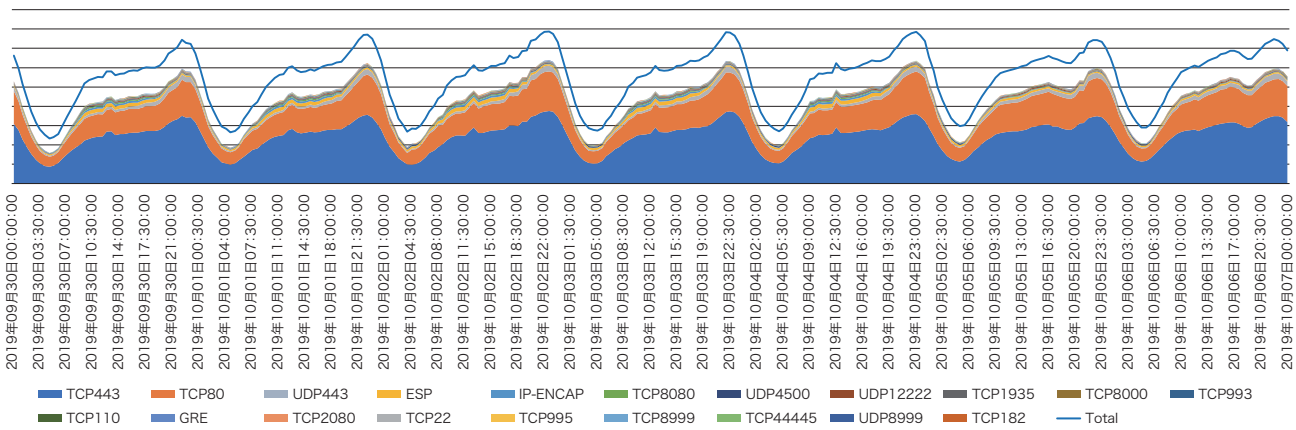

図-9 IPv4トラフィックのProtocol番号と送信元ポート番号で解析したグラフ

まず、IPv6が有効化されている接続の割合ですが、図-10のとおり、有効48%、無効52%とほぼ半々となっています。接続数はほぼ半々ですが、トラフィックについては、IPv4が約8割、IPv6は2割程度になっています。

まず、IPv6が有効になっている端末のOSを調べてみます。図-11のとおり、8割以上がApple iOSで、Androidは14%でした。次に、IPv6が無効になっている端末のOSを見てみます。図-12のとおり、先程と逆転し、Androidが82%、iOSが8%でした。

Androidは、AppleがIPv6を有効化したiOSバージョン11よりもだいぶ早く、2014年のAndroid5のリリースからIPv6をサポートしているはずですが、端末メーカやMNOのポリシーによって、デフォルトではIPv6が無効化されているものが多く、Apple 1社でコントロールしているiOSと比べると、IPv6の接続という観点ではだいぶ差がついてしまったようです。

とはいえ、Apple iOSでもIPv6が無効となっている端末はあり、iOS全体の9.21%でした。推測ですが、iOS 11以降がサポートされない古い機種だと思われます。そして、AndroidでIPv6が有効なものはAndroid全体の14.08%で、最近のSIMロックフリー端末では、IPv6が有効化されたものが多いように見受けられます。

■ まとめ

IPv6のトラフィック量、利用プロトコル、そしてモバイル端末OS別のIPv6接続数について見てきました。IPv4トラフィックの伸びが鈍化する中、IPv6トラフィックは前回同様の伸びを示しており、IPv6の利用が更に進んでいることが分かりました。最近発売される家庭用Wi-Fiルータの多くはIPv6 IPoE対応を謳っていますし、動画配信サービスにおいてもIPv6の有効化の動きがありますので、今後更にIPv6トラフィックは増えていくのではないかと思われます。

モバイルサービスにおいては、IPv6が有効となっている接続が半数近くあり、想像以上にIPv6が有効になっていました。トラフィックはまだまだ全体の2割程度ですので、今後サービス側のさらなるIPv6対応を期待したいと思います。

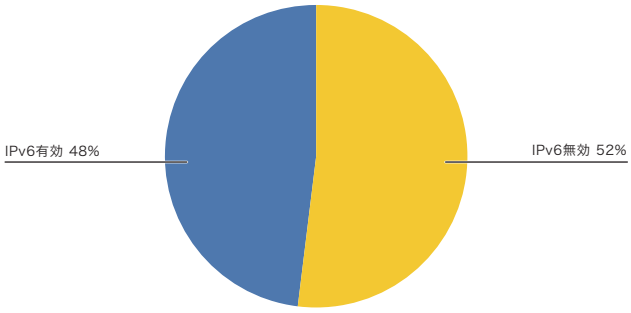

図-10 IPv6が有効化されている接続の割合

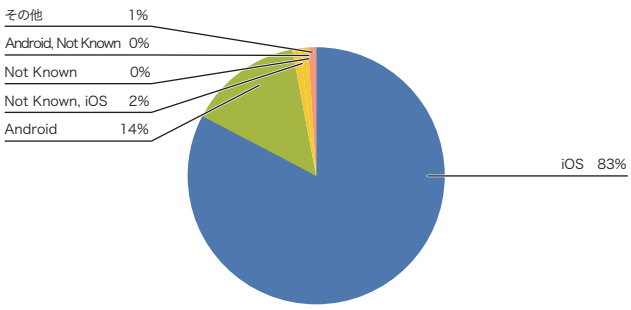

図-11 IPv6が有効になっている端末のOS

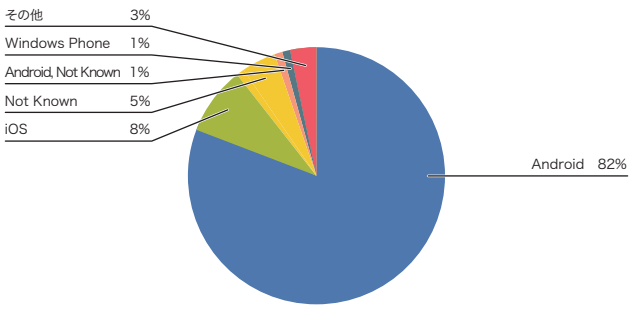

図-12 IPv6が無効になっている端末のOS

Theme 04

モバイル・フレッツと自然災害

本来であればフレッツ及びモバイルの定点観測として傾向の変化を報告するところですが、本レポートのVol.44（https://www.iij.ad.jp/dev/report/iir/044/01.html）の「1. 定期観測レポート」でほぼ同様の内容が報告されていますので、ここでは自然災害が及ぼす影響について考察します。

2019年は自然災害が多く発生した年でした。特に、先日の令和元年台風第19号（以下、台風19号）の際は各地で甚大な被害が発生しました。IIJのバックボーンでも東京-大阪間を接続する基幹回線が複数本、長期間に渡り切断されました。経由する長野県で道路が流され、埋設されていた光ファイバーが物理的に破損したことによるものです。破損した区間が長く、復旧には数週間を要しました。幸い、IIJのバックボーンではこのような災害などによる障害を想定してネットワークを設計しており、東京-大阪間においては太平洋側、日本海側、内陸にルート分散しているため、ユーザの通信に実影響が及ぶことはありませんでした。

一方、モバイル及びフレッツといったユーザの利用に直結するアクセスサービスでは台風19号の影響が如実に見られました。それについてデータを見ながら考察します。なお、ここでのフレッツはPPPoEのみを指すものとします。

図-13から図-24は、台風19号が本州を通過した2019年10月12日前後及び対比のため1週間前の2019年10月5日前後の接続数とトラフィック量（アップロードとダウンロードの合計）の

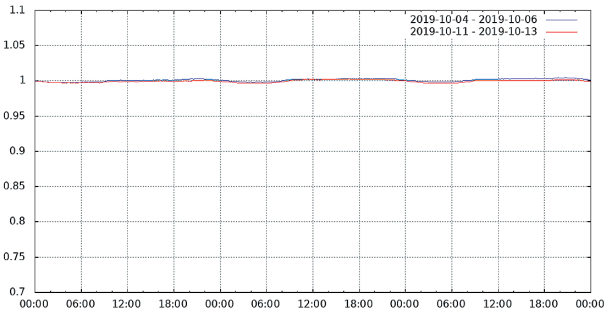

図-13 フレッツ大阪エリアの接続数

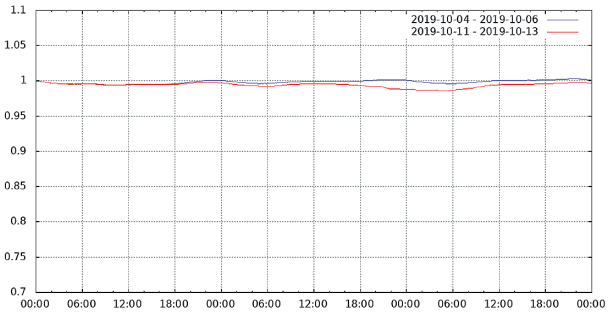

図-15 フレッツ東京エリアの接続数

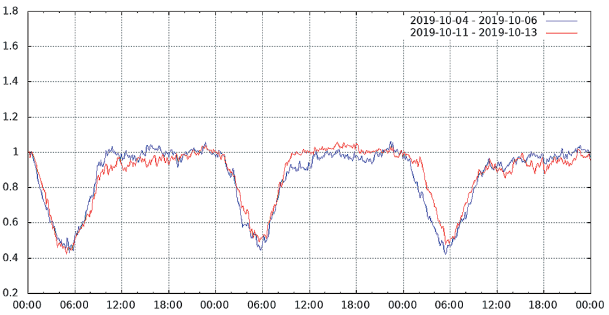

図-14 フレッツ大阪エリアのトラフィック量

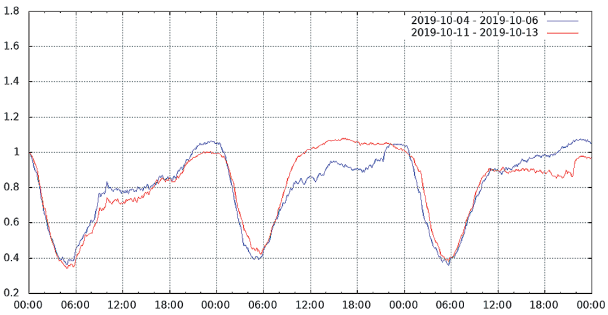

図-16 フレッツ東京エリアのトラフィック量

推移を表したグラフです。いずれも、金曜日(2019年10月4日、2019年10月11日)00:00の値を1として正規化しています。

まずは接続数に着目してみます。現在、フレッツの利用者の多くはブロードバンドルータやホームゲートウェイを介した常時接続のため、通常は1日を通じて接続数の変化はほとんどありません。実際、台風19号の影響がほとんどなかったフレッツ大阪エリアでは、期間中の接続数の変化はごくわずかでした。しかし、影響の大きかったフレッツ千葉、福島、宮城エリアでは2019年10月12日から2019年10月13日にかけて接続数の低下が見られます。特にフレッツ千葉エリアでは大幅に低下しています。これらの地域では停電が多く発生しており、各家庭にあるブロードバンドルータやホームゲートウェイが停電により停止したことによるものと考えられます。一方、モバイルについてはMVNO設備の特性上、端末がどこに在圏しているかを把握することができないため全国での集計となりますが、2019年10

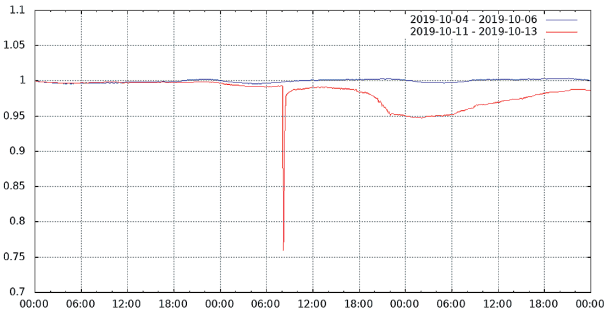

図-17 フレッツ千葉エリアの接続数

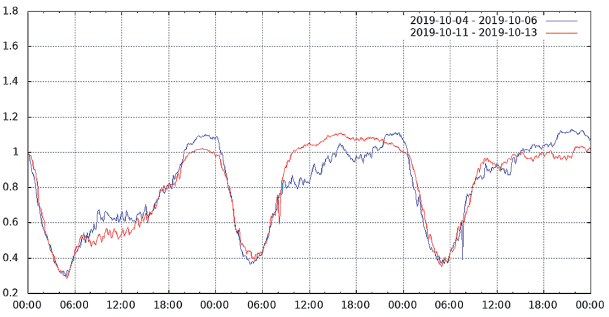

図-18 フレッツ千葉エリアのトラフィック量

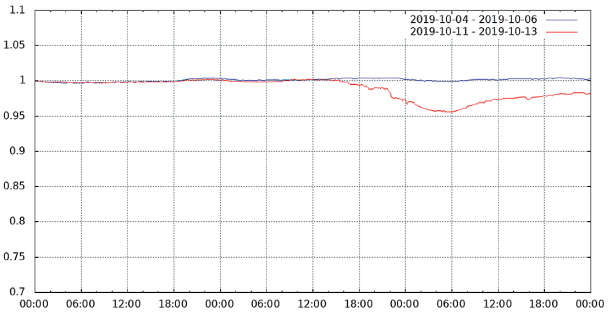

図-19 フレッツ福島エリアの接続数

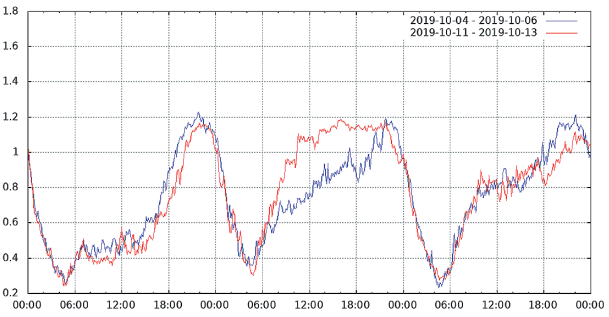

図-20 フレッツ福島エリアのトラフィック量

月12日から2019年10月13日にかけて接続数の低下が見られます。MNOの設備にも被害が及んだ影響、もしくは外出を控えてモバイルを利用しなかったことによる影響と考えられます。

次にトラフィック量について見てみます。トラフィック量は東京以北の各フレッツエリアで2019年10月12日の日中に増加し、逆にモバイルでは大きく低下しています。週末でしたが、荒天のため外出を避け、自宅でモバイルではなくフレッツを使いインターネットを利用するユーザが多かった影響と考えられます。

今やインターネットは電気、ガス、水道に次ぐ重要な社会インフラとなっており、自然災害時こそ真価を発揮します。今後もIIJでは自然災害による影響も十分考慮しながらバックボーンを設計し、いつでも安心して使えるインターネット基盤を提供していきます。

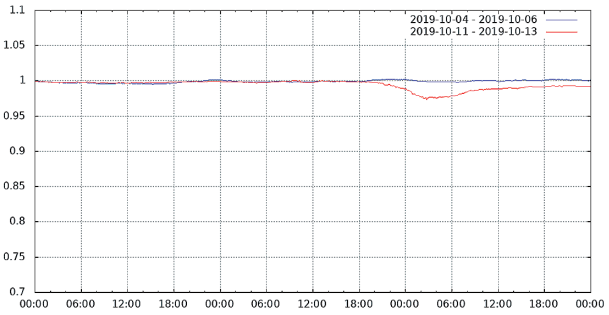

図-21 フレッツ宮城エリアの接続数

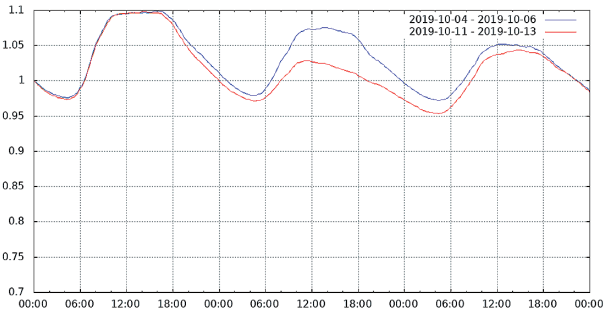

図-23 モバイルの接続数

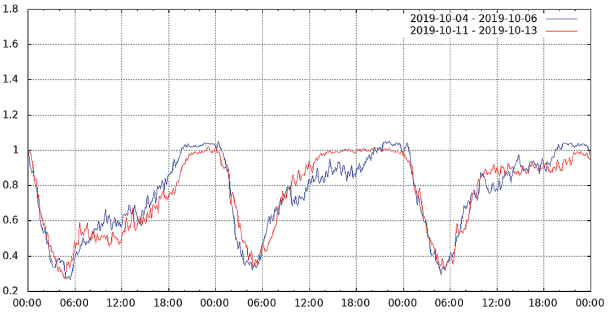

図-22 フレッツ宮城エリアのトラフィック量

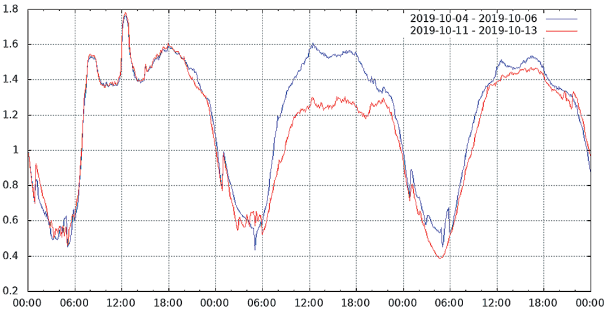

図-24 モバイルのトラフィック量

Theme 05

IIJバックボーンの歴史

2018年のこの記事で、IIJのバックボーンの総トラフィックが10年で10倍以上になっていることを紹介しました。今回はIIJのバックボーンの歴史、特に回線品目について振り返ってみたいと思います。

IIJは特別第二種電気通信事業者としてISP事業をスタートしました。そのためサービス開始当初から第一種電気通信事業者が設置した通信回線を借りて通信網を構築し、サービスを提供してきました。第一種/第二種という区分が廃止された現在でも、この点は基本的に変わっていません。

今、手元に1998年当時のバックボーンマップが残っています。当時利用していた回線は、45MbpsのDS-3が大部分で一部に155MbpsのSTM-1が利用されていました。1998年以降、刻々とバックボーンが増強されていく様子がバックボーンマップに残っています。しばらくはおおむね2年おきに4倍の帯域の回線/ルータのインタフェースがリリースされる状況が続きます。2000年に600Mbps（STM-4）、2002年に2.4Gbps（STM-16）、2004年に9.6Gbps（STM-64）と次々に拡張されていきます。今振り返ってみればトラフィックの増加に合わせて回線も太くなっていく幸せな時代でした。もっと大きな規模のISPは事情が異なったかも知れませんが、IIJの規模ではネットワークのトポロジーを大きく変えずに回線を入れ替えていけばトラフィックの増加におおむね追従できていました。

次のグラフはバックボーンマップから、日米区間のバックボーンの帯域と、バックボーンの中で利用できた最も太い回線品目をプロットしたものです。1998年から2004年のSTM-64を利用開始するまでは回線は帯域を順調に広げています。しかし2005年以降、2014年までおよそ9年もの間、STM-64以上の回線が普及しませんでした。STM-256が今一つ普及し

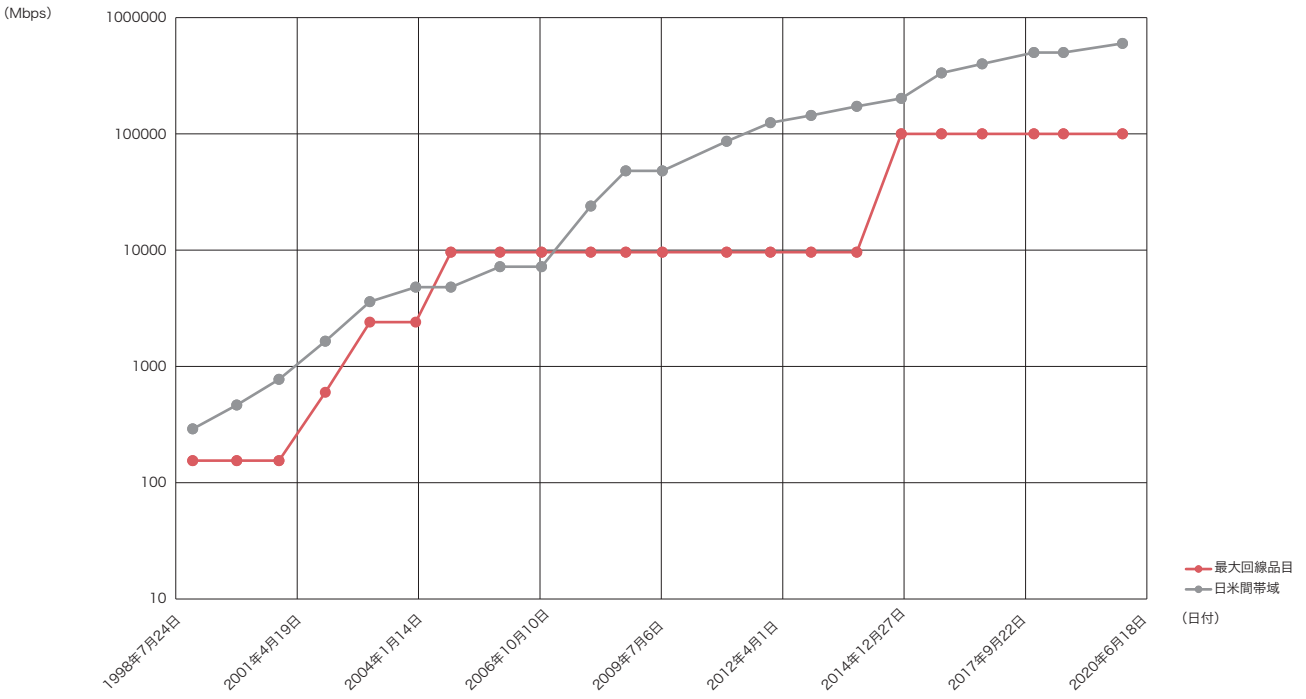

図-25 回線品目と日米間帯域

なかったり、その上の100Gが技術的に困難で開発/規格化が大きく遅れたなど、いろいろと事情はありました。しかし、その間もトラフィックは増え続け、ついには200Gbpsの帯域、STM-64の回線を20本確保しなければならないところまで至ってしまいました。(先のグラフの縦軸は対数ですよ!!)

このため、この間のIIJでのバックボーン構成は非常に大きく変化しました。回線の利用効率を上げるため、ECMP (Equal Cost Multi Path)が効きやすいようにネットワークトポロジーを変更したり、MPLSで回線を仮想化するレイヤーを作りIPバックボーンの構成の自由度を上げたりといったようにです。

2014年に100G Ethernetが普及すると状況はいったん落ち着きました。IIJも昨年、主要なバックボーン区間の100G化が完了し、トラフィック増に応じてリーフの100G化を行っているところです。一方で、この12月には日米間に6本目の100G回線が上がる予定になっています。100Gも普及を始めてから5年が経ち、そろそろ一段太い回線が欲しくなってきました。400Gの足音が少しずつ大きくなってきているようです。

IIJでは引き続き、最新の技術動向を追いかけながら効率の良い快適なバックボーンを拡大していきます。

執筆者:

1.BGP・経路数
倉橋 智彦（くらはし ともひこ）
IIJ 基盤エンジニアリング本部 運用技術部 技術開発課

2.DNSクエリ解析
松崎 吉伸（まつざき よしのぶ）
IIJ 基盤エンジニアリング本部 運用技術部 技術開発課

3.IPv6
佐々木 泰介（ささき たいすけ）
IIJ 基盤エンジニアリング本部 ネットワーク技術部 副部長

4.モバイル・フレッツと自然災害
堀 高房（ほり たかふさ）
IIJ 基盤エンジニアリング本部 ネットワーク技術部 ネットワーク技術課長

5.IIJバックボーンの歴史
篠井 隆典（ささい たかのり）
IIJ 基盤エンジニアリング本部 ネットワーク技術部 バックボーン技術課長

Linuxのフォレンジック向けメモリイメージ取得

2.1 Linuxメモリダンプツール

インシデントレスポンスやフォレンジックでメモリ解析を行う場合、私たちはVolatilityを使用しています。本レポートのVol.32の「1.4.1 Volatility Frameworkプロファイルの生成」ではVolatilityのLinux用プロファイルを作成する手順を解説しました[*1]。

今回はVolatilityで解析を行うLinuxのメモリイメージをどのように取得すれば良いか解説します。Linuxのメモリイメージを取得するツールはいくつかありますが、今回はLiME[*2]とcrash[*3]というツールを紹介します。また、ディスクフォレンジックに極力影響がないようなメモリイメージ取得方法も紹介します。動作環境はCentOS 7.7-1908を想定しています。メモリイメージを取得するツールは、これら以外にもLinpmem[*4]がありますが、筆者の検証環境ではうまく動作しなかったため、今回は紹介を見送ることにしました。

なお、本稿では調査が行われる側のコンピュータを「解析対象ホスト」、調査を行う側のコンピュータを「調査用ホスト」と記述します。解析対象ホストのデータを極力、変更しないようにするため、解析ツールのコンパイルなどは解析対象のホストとは別に用意し、そのホスト上で行ってください。

2.2 LiMEとは

LiMEはLinux Memory Extractorの略でVolatilityがメモリイメージの取得に推奨している[*5]ツールです。LiMEはLinuxのカーネルモジュールとして動作するため、解析対象ホストで動作しているカーネルのバージョンに合わせてLiMEをコンパイルする必要があります。

2.3 LiMEのコンパイル

LiMEのコンパイルを行うには、最初に図-1のようにgitコマンドでLiMEのソースコードを取得するか、GitHubのLiMEのページからzipファイルをダウンロードして適当なディレクトリに展開します。ソースコードを展開したディレクトリ内の「src」ディレクトリにMakefileがあるので、このディレクトリでmakeコマンドを実行すればLiMEモジュールが生成されます（図-2の赤枠で囲ったファイル）。LiMEのコンパイルには、カーネルモジュールのコンパイルに必要なパッケージ（kernel-develやgccなど）が必要になるため、makeコマンド実行時にエラー

```
$ git clone https://github.com/504ensicsLabs/LiME.git
```

図-1 LiMEのGitリポジトリをクローン

```
$ cd LiME/src
$ make
$ ls
disk.c  lime-3.10.0-1062.el7.x86_64.ko  lime.o    Makefile.sample  tcp.o
disk.o  lime.h                          main.c    modules.order
hash.c  lime.mod.c                      main.o    Module.symvers
hash.o  lime.mod.o                      Makefile  tcp.c
```

図-2 LiMEモジュールをコンパイル

```
$ sudo yum install kernel-devel gcc
```

図-3 kernel-develパッケージをインストール

*1 Internet Infrastructure Review（IIR）Vol.32 1.4.1 Volatility Frameworkプロファイルの生成（https://www.iij.ad.jp/dev/report/iir/032/01_04.html）。
*2 LiME（https://github.com/504ensicsLabs/LiME）。
*3 crash（https://people.redhat.com/anderson/）。
*4 Velocidex/c-aff4（https://github.com/Velocidex/c-aff4/releases）。
*5 Linux · volatilityfoundation/volatility Wiki（https://github.com/volatilityfoundation/volatility/wiki/Linux#acquiring-memory）。

が発生する場合は図-3のように関連するパッケージをインストールしてみてください。

なお、kernel-develパッケージのインストール時にバージョンを指定しない場合、最新バージョンのパッケージがインストールされます。解析対象ホストで異なるバージョンのカーネルが動作している場合、図-4のようにyumコマンドで検索を行って適切なバージョンのkernel-develパッケージをインストールし、LiMEモジュールのコンパイルを行ってください。この際、makeコマンドにKVERオプションが必要になります。

更にOSのバージョンが異なる場合には、図-5のように該当のkernel-develパッケージを個別にダウンロードし、cpioコマンドでパッケージを展開後、KVERとKDIRオプションを指定したmakeコマンドを実行することで、該当のバージョン用のLiMEモジュールを作成することができます。KVERはカーネルバージョンを、KDIRはkernel-develパッケージを展開したディレクトリをそれぞれ指定します。その他、必要なパッケージがあればインストールします（筆者の環境ではelfutils-libelf-develをインストールする必要がありました）。このように作成したLiMEモジュールはCentOS 8（1905）で動作することを確認しました。

2.4 外部ドライブへのメモリダンプ

メモリイメージを取得する際、解析対象ホストのディスクに極力書き込みを行わないようにする必要があります。特にメモリイメージは数GB以上のファイルになることがほとんどであり、解析対象ホストのディスクに書き込みを行った場合、多くの未使用領域を上書きすることになるため、ディスクフォレンジックによる調査に大きな影響を及ぼす可能性があります。

そのため、解析対象ホストに物理的アクセスができるのであれば、LiMEモジュールをコピーしたUSBメモリやモバイルSSDを解析対象ホストに接続し、図-6のようにinsmodコマンドでカーネルにLiMEモジュールをロードすることで、解析対象ホストのディスクにほとんど書き込みを行わずにメモリイメージを取得することができます。図-6のダブルクォーテーションで囲っている部分はLiMEモジュールのオプションです。この例の場合、/mediaにマウントされたUSBメモリにlimeフォー

```
$ yum --showduplicates search kernel-devel
$ sudo yum install kernel-devel-3.10.0-1062.1.2.el7.x86_64
$ make KVER=3.10.0-1062.1.2.el7.x86_64
```

図-4 カーネルバージョンを指定したLiMEのコンパイル（1）

```
$ curl -OL http://ftp.iij.ad.jp/pub/linux/centos/8.0.1905/BaseOS/x86_64/os/Packages/kernel-devel-4.18.0-80.11.2.el8_0.x86_64.rpm
$ rpm2cpio ./kernel-devel-4.18.0-80.11.2.el8_0.x86_64.rpm | cpio -id
$ sudo yum install elfutils-libelf-devel
$ make KVER=4.18.0-80.11.2.el8_0.x86_64 KDIR=~/src/usr/src/kernels/4.18.0-80.11.2.el8_0.x86_64/
```

図-5 カーネルバージョンを指定したLiMEのコンパイル（2）

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=/media/centos77.mem format=lime"
```

図-6 LiMEモジュールのロード

マットのダンプファイルを/media/centos77.memというファイル名で保存します。

LiMEモジュールがロードされるとメモリダンプが開始されますが、メモリダンプが完了するまでコマンドプロンプトは返ってきません。モジュールをロードした後に操作ができなくなっても焦らずにダンプが終了するまで待ってください。特に最近のサーバではメモリ容量が大きいため時間がかかることが予想されます。なお、メモリダンプが完了してもLiMEモジュールはロードされたままなので、図-7のようにrmmodコマンドでモジュールをアンロードします。

2.5 ネットワーク経由でのメモリダンプ

上記のように解析対象ホストに物理的にアクセスできない場合（例えば、解析対象ホストが遠隔地にあるため物理的なアクセスが容易ではない、など）、または大容量のUSBメモリがない場合、他のツールと組み合わせてネットワーク経由でメモリイメージを取得します。ここでは、Netcat、NFS、SSHの3種類のツールと組み合わせる方法を紹介します。また、調査用ホストのIPアドレスを192.168.232.131、解析対象ホストのIPアドレスを192.168.232.132とします。

■ Netcat（1）

LiMEモジュールはネットワーク経由でメモリダンプする機能を持っており、この機能とNetcatコマンドを組み合わせてネットワーク経由でメモリイメージを取得します。調査用ホストにNetcatコマンドをインストールし、図-8と図-9のようにコマンドを実行すると、調査用ホストからLiMEモジュールがLISTENしているポート（4444/tcp）に接続し、メモリイメージのデータを取得します。

■ Netcat（2）

上記の手順ではメモリ容量と同じだけのデータをネットワーク経由で受信することになります。調査用ホストがサーバの場合、数十ギガバイト以上のデータになることもあるため、可能な限り圧縮したいところです。解析対象ホストにもNetcatが

```
$ lsmod | grep lime
$ sudo rmmod lime
```

図-7 LiMEモジュールのアンロード

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=tcp:4444 format=lime"
```

図-8 LiMEで4444/tcpをListenする（解析対象ホスト）

```
$ nc 192.168.232.132 4444 > centos77.mem
```

図-9 Netcatでネットワーク経由でメモリイメージを取得（調査用ホスト）

```
$ nc -l 5555 > memorydump.lime.gz
```

図-10 調査用ホストで実行するコマンド

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=tcp:4444 format=lime"
別のログインセッションに切り替えて、以下のコマンドを実行する。
$ nc localhost 4444 | gzip -c | nc 192.168.232.131 5555
```

図-11 解析対象ホストで実行するコマンド

```
$ sudo yum install nfs-utils
$ sudo mkdir /mnt/nfsserv/
$ chown -R nfsnobody:nfsnobody /mnt/nfsserv/
$ sudo vi /etc/exports
$ sudo systemctl start nfs.service
$ sudo systemctl status nfs.service
```

図-12 NFSサーバ（調査用ホスト）の設定

```
/mnt/nfsserv/ 192.168.232.132(rw,all_squash)
```

図-13 調査用ホストのNFSエクスポート設定（/etc/exports）

インストールされていれば、図-10と図-11のようなコマンドを実行することでメモリイメージをgzipで圧縮しながら調査用ホストに転送することが可能です。先ほど既述したように、LiMEモジュールを読み込むとコマンドプロンプトは返ってきませんので、「nc」以降のコマンドは別のログインセッションから実行する必要があります。

この例では、まず調査用ホストでNetcatを使用して5555/tcpでLISTENを行います。次に解析対象ホストでNetcatを使ってLiMEモジュールがLISTENしている4444/tcpに接続し、メモリイメージを取得後、gzipで圧縮し、更にNetcat経由で調査用ホストに転送します。

■ NFS

解析対象ホストがNFSボリュームをマウントすることが可能である場合、解析対象ホストがアクセス可能なネットワーク上に調査用ホストを用意します。そして、調査用ホストでNFSボリュームを読み書き可能な設定でエクスポートします。このNFSボリュームにLiMEをコピーしておき、解析対象ホストからNFSマウントすれば、解析対象ホストにファイルを作成することなくメモリイメージを取得することが可能です（図-12、図-13、図-14、図-15）。

■ SSH

NetcatやNFSが使えない場合、代わりにSSHを使用することができます。図-16のようなコマンドを実行することで調査用ホストにメモリイメージをSSH経由で転送することができます。ただし、この手法が使えるのはbashのみになります。また、Netcat（2）の場合と同様に、exec以降のコマンドは別のログインセッションから実行する必要があります。

2.6 crashとは

crashコマンドはLinuxのメモリイメージを解析するためのツールです。解析を行うのが主目的のツールですが、メモリダンプを行うためのモジュールも使用することができます。ただし、crashのメモリダンプモジュールはRPMパッケージ

```
$ sudo firewall-cmd --permanent --add-service=nfs
$ sudo firewall-cmd --reload
$ sudo exportfs -v
```

図-14 調査用ホストのファイアウォール設定とNFSエクスポート確認

```
$ sudo mount -t nfs 192.168.232.131:/mnt/nfsserv/ /mnt/
$ sudo insmod /mnt/lime-3.10.0-1062.el7.x86_64.ko "path=/mnt/centos77.mem format=lime"
```

図-15 NFSマウントの実行とメモリイメージ取得（解析対象ホスト）

```
$ sudo insmod /media/lime-3.10.0-1062.el7.x86_64.ko "path=tcp:4444 format=lime"
別のログインセッションに切り替えて、以下のコマンドを実行する。
$ exec 5<>/dev/tcp/127.0.0.1/4444; cat <&5 | ssh -c user@192.168.232.131 'cat > centos77.mem'
```

図-16 解析対象ホストで実行するコマンド

が用意されていないため、ソースパッケージからコンパイルする必要があります(図-17)。また、crashコマンドの動作にはデバッグシンボル付きのカーネルが必要になるため、デバッグ用のカーネルパッケージをインストールします(図-18)。次に、図-19の3つのファイルをUSBメモリ内の同じディレクトリにコピーします。解析対象ホストでは、図-20のようにcrashを実行してメモリイメージを取得します。

なお、解析用ホストと解析対象ホストのカーネルバージョンが異なる場合は、図-4を参考に適切なバージョンのkernel-debuginfoパッケージを検索してください(yum --showduplicates search)。更に図-5を参考にパッケージをダウンロード及び展開(curl、rpm2cpio、cpioコマンド)し、vmlinuxファイルをコピーしてください。crashコマンドもOSバージョンに合わせたバージョンを使った方が良いでしょう(例えば、CentOS 8.0ではcrash-7.2.3-18が提供されています)。

2.7 メモリイメージの解析

Volatility[*6]でLinuxのメモリイメージを解析するには、Linuxカーネルのバージョンに応じたプロファイルが必要になります。冒頭で述べたように、Volatilityプロファイルの作成手順はIIR Vol.32[*1]で取り上げましたので、手順が分からない方は参考にしてください。

また、本稿の執筆中にLiMEモジュールとVolatility用Linuxプロファイルを自動生成して公開するBitbucketリポジトリをLorenzo Martínez氏が公開しました[*7]。このリポジトリは新しいバージョンのLinuxカーネルパッケージがリリースされるたびに更新されます。ただし、自動生成する対象のOSはCentOS 5、6、7、8及びUbuntu 14.04 LTS、16.04 LTS、18.04 LTSになります。リポジトリのWebページでカーネルバージョンでフィルタリングを行えば、該当するLiMEモジュールとVolalityプロファイルをダウンロードすることができます。

```
$ sudo yum install crash crash-devel
$ yumdownloader --source crash
$ rpm -ivh crash-7.2.3-10.el7.src.rpm
$ cd rpmbuild/SPECS
$ rpmbuild -bp crash.spec
$ cd ../BUILD/crash-7.2.3
$ make extensions
```

図-17 crashのソースパッケージインストールとモジュールのコンパイル

```
・/usr/bin/crash
・rpmbuild/BUILD/crash-7.2.3/extensions/snap.so
・/usr/lib/debug/usr/lib/modules/3.10.0-1062.el7.x86_64/vmlinux
```

図-19 メモリダンプに必要なファイル

```
$ sudo yum install --enablerepo=base-debuginfo kernel-debuginfo-3.10.0-1062.el7.x86_64
```

図-18 デバッグシンボル付きカーネルインストール

*6 The Volatility Foundation - Open Source Memory Forensics(https://www.volatilityfoundation.org/)。
*7 Lorenzo Martínez氏のツイート(https://twitter.com/lawwait/status/1181469996821700609)。

なお、x86_64以外のアーキテクチャには対応していないため、自動生成の対象ではないLinuxディストリビューションやアーキテクチャを使っている場合は自身でそれらを用意する必要があります。また、Linuxカーネルをカスタマイズして使用している場合も同様です。crashを使うのであれば、自身でデバッグシンボル付きのカスタマイズカーネルも用意する必要があります。

2.8 メモリイメージ取得時のTips

Linuxカーネル2.4からtmpfsというファイルシステムを使うことができます。tmpfs内のデータはメモリ上のみに保持され、ホストのシャットダウンや再起動を行うと保存内容が消えるため、通常はテンポラリディレクトリなどファイルが消えても問題ない用途に使用されます。しかし、この特性を利用して、攻撃者はアンチディスクフォレンジックのためにこの領域をファイル置き場として使用することが確認されています。

そのため、Volatilityにはlinux_tmpfsというLinux専用のコマンドが用意されています。これは、tmpfs内のファイルを復元するコマンドです。図-21では、/home/user/tmpディレクトリにマウントされたtmpfsのファイルを復元しています。コマンドの結果、「tmpfs_example.txt」というファイルが復元され、ファイル内容が「hello!!」であることが確認できます。

しかし、メモリ空き容量が少なくなるとtmpfsの内容はスワップアウトされてしまうため、メモリダンプを行ってもtmpfs内のデータを復元することはできません（図-22）。このような場合の対策として一時的にスワップを無効化することが考えられます。つまり、スワップアウトしたデータを強制的にスワップインさせるということです。ただし、この操作を行うことができるか否かは、解析対象ホストのメモリ使用状況に依存します。一時的にメモリ使用量が上がったことが原因でスワップアウトし、その後、メモリが解放されたような状況であればス

ファイルをコピーしたディレクトリに移動後、以下のコマンドを実行する。

```
$ sudo ./crash ./vmlinux
（以降、crashコマンドのプロンプト）
extend ./snap.so
snap centos77.mem
```

図-20 メモリイメージの取得

```
$ hexdump -C ~/vol_output/swapout/tmpfs_example.txt
00000000  00 00 00 00 00 00 00 00                           |........|
00000008
```

図-22 スワップアウトによりtmpfs内のデータ復元に失敗した例

```
$ python2 ./vol.py --profile=LinuxCentOS77x64 -f ~/tmpfs_swapoff.mem linux_tmpfs -L
Volatility Foundation Volatility Framework 2.6.1
1 -> /sys/fs/cgroup
2 -> /run
3 -> /home/user/tmp
4 -> /dev/shm

$ python2 ./vol.py --profile=LinuxCentOS77x64 -f ~/tmpfs_swapoff.mem linux_tmpfs -S 3 -D ~/vol_output/
$ hexdump -C ~/vol_output/tmpfs_example.txt
00000000  68 65 6c 6c 6f 21 21 0a                           |hello!!.|
00000008
```

図-21 tmpfsに保存されたファイルの復元と内容の確認

ワップ無効化ができる可能性があります。例えば、図-23のようにSwapのusedの値がMemのFreeよりも小さい場合が該当します。スワップ無効化後に取得したメモリイメージを解析すると、図-24のようにtmpfsのデータが復元できることが確認できます。

ただし、この方法はメモリの未使用領域にあるデータを上書きしてしまいます。未使用領域にも有用なデータが残っている可能性があるため、スワップ無効化の前後で1回ずつメモリダンプを行うのが、Linuxのメモリフォレンジックではベターな方法です。

2.9 Volatility 3

Volatilityは長い間バージョン2系が使われてきましたが、本稿執筆中にVolatility 3のパブリックベータ版がリリースされました[*8]。すべての形式のメモリイメージで確認したわけではありませんが、筆者の環境ではWindowsのRAW形式のメモリイメージとLiMEで取得したCentOSのメモリイメージは解析することができました。大きな変更点として、Volatility 2では必須であったプロファイルを指定するオプションがなくなりました。代わりに、Volatility 3では解析対象のメモリイメージからOSの種類とバージョンを推測し、適切なシンボルテーブルを参照します。例えば、Windowsのメモリイメージを読み込ませると、マイクロソフトからPDBファイルを自動的にダウンロードおよび解析してシンボル情報を参照します。しかし、macOSとLinuxのシンボルテーブルについては、Volatility 2と同様に、事前に用意する必要があります。Volatilityの開発元が用意したシンボルテーブルを使用することもできますが、WindowsやmacOSと比べ、Linux向けのシンボルテーブルは情報が不足しているため、ほとんどの場合、ユーザが用意する必要があります。

シンボルテーブルを作成するには、dwarf2json[*9]というツールを使用します。dwarf2jsonはGo言語で書かれているため、まず、golangパッケージをインストールし、dwarf2jsonをビルドします。また、シンボル情報が付与されたLinuxカーネルも必要であるため、kernel-debuginfoパッケージをインストールします。シンボル付きのLinuxカーネルを指定してdwarf2jsonを実行すると、シンボルテーブルが作成されます

```
$ free
              total        used        free      shared  buff/cache   available
Mem:        1863248       75920      307192         768     1480136     1591860
Swap:       2097148       56776     2040372
$ sudo swapoff -a
$ free
              total        used        free      shared  buff/cache   available
Mem:        1863248      118804      247652        9800     1496792     1539936
Swap:             0           0           0
（メモリダンプ後、再度、スワップを有効化）
$ sudo swapon -a
```

図-23 メモリ使用状況の確認とスワップ無効化および有効化

```
$ hexdump -C ~/vol_output/swapoff/tmpfs_example.txt
00000000  68 65 6c 6c 6f 21 21 0a                           |hello!!.|
00000008
```

図-24 スワップ無効化後のメモリイメージから復元したtmpfsデータ

```
$ sudo yum install epel-release
$ sudo yum install golang
$ git clone https://github.com/volatilityfoundation/dwarf2json.git
$ cd dwarf2json
$ go build
$ sudo yum install --enablerepo=base-debuginfo kernel-debuginfo-3.10.0-1062.1.2.el7.x86_64
$ ./dwarf2json linux --elf /usr/lib/debug/usr/lib/modules/3.10.0-1062.1.2.el7.x86_64/vmlinux > centos77-3.10.0-1062.1.2.el7.x86_64.json
$ xz -z centos77-3.10.0-1062.1.2.el7.x86_64.json
$ wget https://downloads.volatilityfoundation.org/volatility3/symbols/linux.zip
$ zip ./linux.zip ./centos77-3.10.0-1062.1.2.el7.x86_64.json.xz
```

図-25 dwarf2jsonのビルドとシンボルテーブルの作成

*8 Volatility Labs：Announcing the Volatility 3 Public Beta!（https://volatility-labs.blogspot.com/2019/10/announcing-volatility-3-public-beta.html）。
*9 dwarf2json（https://github.com/volatilityfoundation/dwarf2json）。

ので、これを開発元が配布しているシンボルテーブルが納められたファイル(linux.zip)に追加します。具体的なコマンドは図-25を参照してください。作成したシンボルテーブルはVolatility 3の所定のディレクトリにコピーします(図-26)。

Volatility 3は、「python3 vol.py -f <メモリイメージファイル> <プラグイン>」のような形式のコマンドラインで実行します。図-27はCentOS 7.7のメモリイメージをpstreeプラグインで解析した際の実行結果です。プロセスのネストを表す記号が「*」となっており、Volatility 2から若干フォーマットが変更されています。また、実行するプラグインの指定方法も変更されています。pstreeプラグインの場合、Volatility 2では「linux_pstree」と指定していましたが、Volatility 3では「linux.pstree.PsTree」と指定します。使用できるプラグインの一覧は「python3 vol.py -h」を実行すると分かります。

正常に動作することが確認できた一方で、いくつかの不具合も確認しました。先ほど書いたように、OSの種類とバージョンはコマンドラインで指定したメモリイメージの内容から推測されますが、メモリイメージの状態によっては、Linuxカーネルバージョンが正しく認識されず解析が失敗してしまう場合がありました。また、Windowsのメモリイメージではダウンロードしたpdbファイルの解析に失敗してしまい、メモリイメージの解析まで処理が進まない場合がありました。

Volatilityの開発チームはVolatility 3の正式バージョンを2020年8月に公開するとアナウンスしています。Volatility 2はその1年後の2021年8月までサポートを続けるとしていますが、今後、主流となるVolatility 3のリリースに備えて、早めに新しい使い方や設定方法などを確認しておくとよいでしょう。

```
$ cd ..
$ sudo yum install python3
$ git clone https://github.com/volatilityfoundation/volatility3.git
$ pip3 install --user pefile yara-python capstone
$ cp ./dwarf2json/linux.zip ./volatility3/volatility/symbols/
```

図-26 Volatility 3のインストールと
作成したシンボルテーブルファイルのコピー

```
$ cd volatility3
$ python3 vol.py -f ~/centos77.mem linux.pstree.PsTree
Volatility 3 Framework 1.0.0-beta.1
Progress:  23.13               Scanning LimeLayer using RegExScanner
PID     PPID    COMM

1       0       systemd
* 833   1       login
** 1695 833     bash
* 841   1       firewalld
* 843   1       NetworkManager
** 992  843     dhclient
* 558   1       systemd-journal
* 593   1       systemd-udevd
* 818   1       dbus-daemon
* 1171  1       sshd
** 1720 1171    sshd
*** 1724        1720    sshd
**** 1725       1724    bash
***** 1751      1725    sudo
****** 1753     1751    insmod
* 1172  1       tuned
* 821   1       systemd-logind
* 822   1       irqbalance
* 823   1       polkitd
* 1174  1       rsyslogd
* 1397  1       master
** 1402 1397    pickup
** 1405 1397    qmgr
(以下略)
```

図-27 Linuxメモリイメージに対してpstreeプラグインを実行

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。

December 2019 Vol.45

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。

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