Olympic Destroyerの内部構造を紐解く | 技術者ブログ | 三井物産セキュアディレクション株式会社

2月8日から韓国で開催されている平昌オリンピックが日々世間を賑わせているさなか、平昌オリンピックを狙ったサイバー攻撃に関する情報が水面下で流れています。

2月9日頃から平昌オリンピック組織委員会の内部でシステムトラブルが発生していた問題で、2月11日になり平昌オリンピック組織委員会から一連の障害はサイバー攻撃によるものと明らかにされました。

また、CiscoのTalosはこの攻撃に用いられたとみられるマルウェア「Olympic Destroyer」（オリンピック・デストロイヤー）に関する情報を公開しました（※1）。（なお、Talosは公開時点で当該情報を中程度の信頼度としています）

(※１：Olympic Destroyer Takes Aim At Winter Olympics：http://blog.talosintelligence.com/2018/02/olympic-destroyer.html)

我々は「Olympic Destroyer」とされている検体の一つを入手し、独自に解析を進めた結果、調査時点で新たにいくらかの興味深い解析結果が見えてきたため、以下にマルウェアの全体像と共にまとめて共有します。

「Olympic Destroyer」の構成図

Olympic Destroyerの本体は実行形式（EXE）ファイルであり、自身の内部（リソースセクション）に５つの難読化されたファイルを保有しています。これらのファイルはそれぞれ目的をもったEXEファイルであり、必要に応じて復号・使用されます。

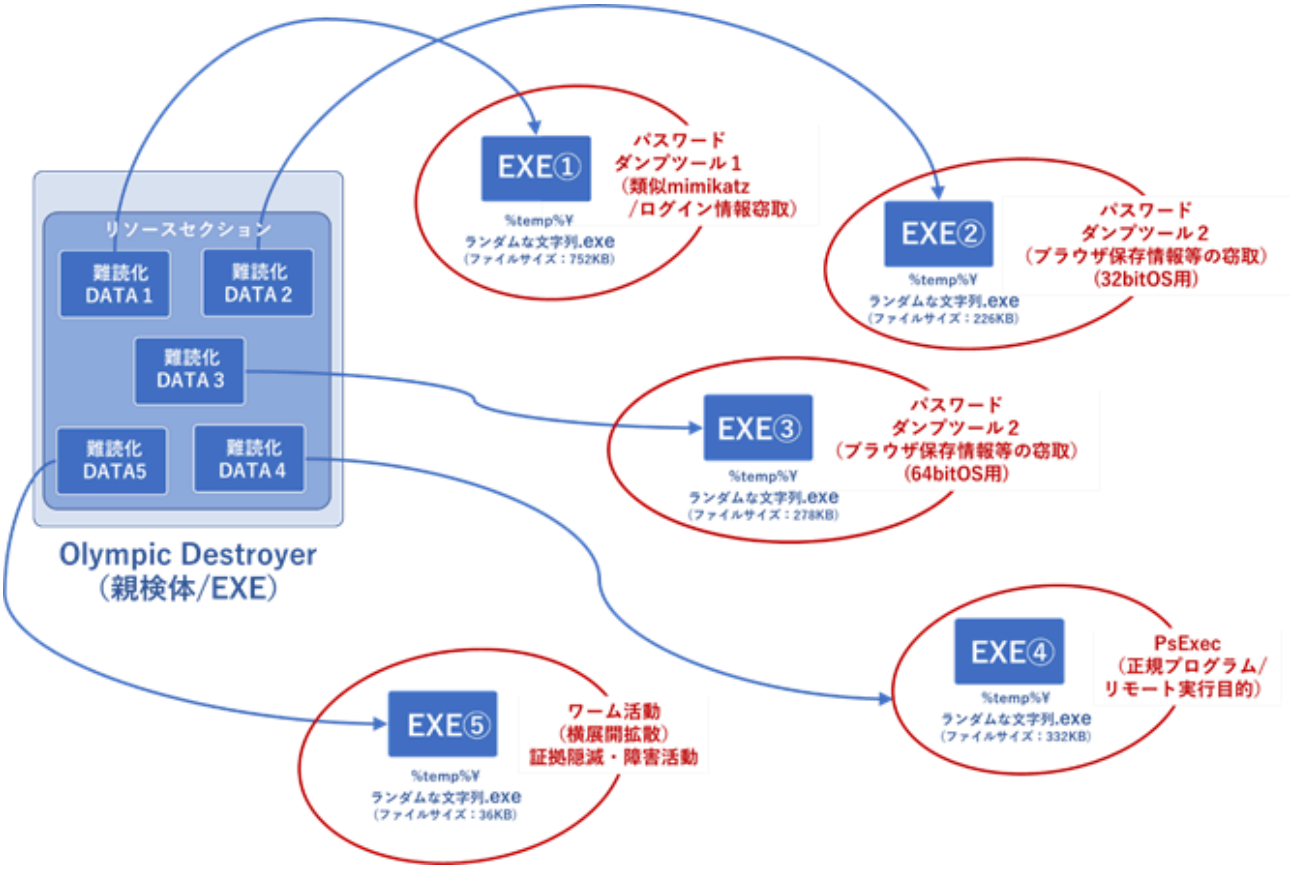


図１　Olympic Destroyerの構成図

リソースファイルの一部には、昨年度話題となった「NotPetya」と同様に、正規プログラムである「PsExec」を横展開時に利用するために難読化して保有しています。

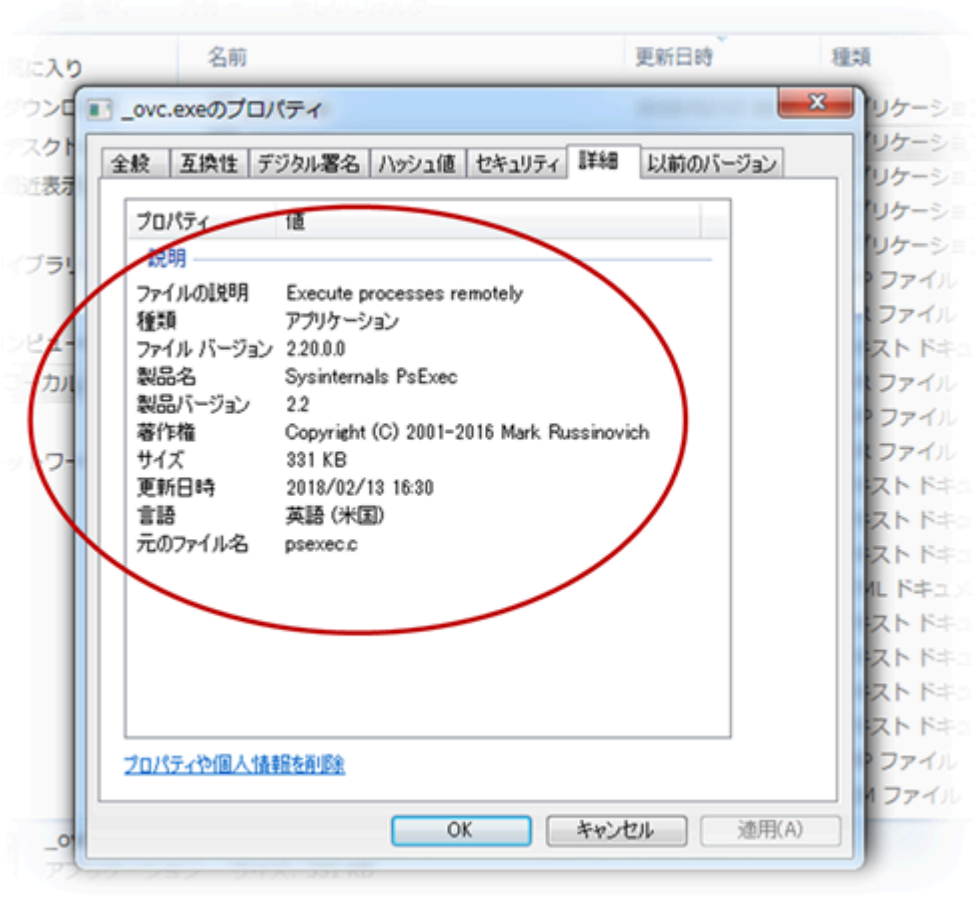


図2　リソースセクションに埋め込まれているPsExec

「Olympic Destroyer」の挙動

Olympic Destroyerは、実行されると、%temp%フォルダに以下のEXEファイルを作成します。

- %temp%\<ランダムな文字列>.exe ・・・（１）パスワードダンプツール①（ログイン情報用）
- %temp%\<ランダムな文字列>.exe ・・・（２）パスワードダンプツール②（ブラウザ情報等用）
- %temp%\<ランダムな文字列>.exe ・・・（３）PsExec（正規プログラム／リモート実行ツール）
- %temp%\<ランダムな文字列>.exe ・・・（４）ワーム活動・証拠隠滅・障害活動

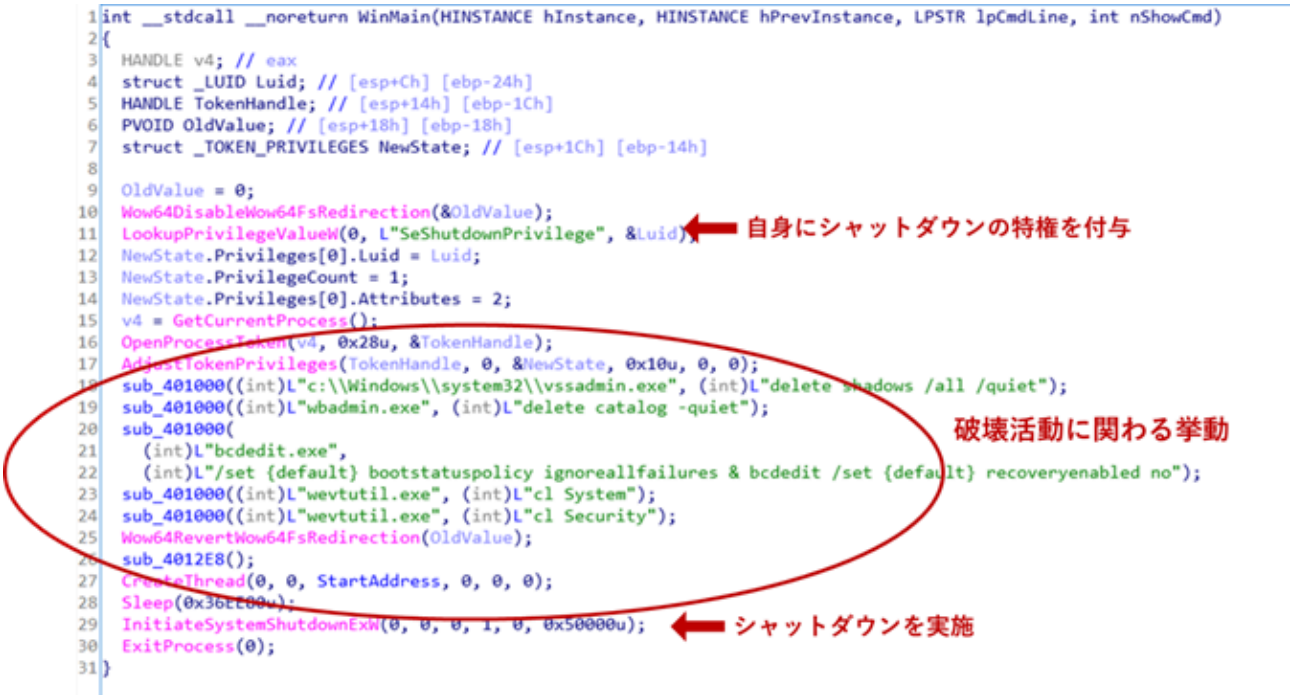
まず（１）と（２）のEXEファイルを実行することで、OSのログイン情報および、ブラウザの保存情報などの各種ログイン情報・パスワード情報を取得します。

次に、（３）と（４）のEXEファイルを利用してネットワークを介したワーム活動と破壊活動を行います。具体的には、GetIPNetTableによりARPテーブル情報を取得し、WMI（”SELECT ds_cn FROM ds_computer”クエリ）を利用して得られた端末一覧情報を取得、それらの宛先に対し、自身が上記でドロップしたPsExecを利用することでリモート先への自身のコピー及びリモート実行を行う事で横展開（ワーム活動）を行います。その際上記で摂取したログイン情報を横展開のアクセス試行に利用します。

（４）のEXEファイルは横展開の動作を終えると、以下の証拠隠滅および破壊活動に関わる動作を行います。

- ボリュームシャドーコピー（システムの復元）の削除
- システムバックアップの削除（wbadminによる）
- OSのスタートアップ修復の無効化
- イベントログ（SYSTEMおよびSECURITY）の削除
- 全てのサービスの無効化
- ファイル共有されているファイルの一部を0で上書き破壊

これらの破壊動作を行った後、強制的に端末をシャットダウンします。




```
int __stdcall __noreturn WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
{
  HANDLE v4; // eax
  struct _LUID Luid; // [esp+Ch] [ebp-24h]
  HANDLE TokenHandle; // [esp+14h] [ebp-1Ch]
  PVOID OldValue; // [esp+18h] [ebp-18h]
  struct _TOKEN_PRIVILEGES NewState; // [esp+1Ch] [ebp-14h]

  OldValue = 0;
  Wow64DisableWow64FsRedirection(&OldValue);
  LookupPrivilegeValueW(0, L"SeShutdownPrivilege", &Luid);
  NewState.Privileges[0].Luid = Luid;
  NewState.PrivilegeCount = 1;
  NewState.Privileges[0].Attributes = 2;
  v4 = GetCurrentProcess();
  OpenProcessToken(v4, 0x28u, &TokenHandle);
  AdjustTokenPrivileges(TokenHandle, 0, &NewState, 0x10u, 0, 0);
  sub_401000((int)L"c:\\Windows\\system32\\vssadmin.exe", (int)L"delete shadows /all /quiet");
  sub_401000((int)L"wbadmin.exe", (int)L"delete catalog -quiet");
  sub_401000(
    (int)L"bcdedit.exe",
    (int)L"/set {default} bootstatuspolicy ignoreallfailures & bcdedit /set {default} recoveryenabled no");
  sub_401000((int)L"wevtutil.exe", (int)L"cl System");
  sub_401000((int)L"wevtutil.exe", (int)L"cl Security");
  Wow64RevertWow64FsRedirection(OldValue);
  sub_4012E8();
  CreateThread(0, 0, StartAddress, 0, 0, 0);
  Sleep(0x36EE80u);
  InitiateSystemShutdownExW(0, 0, 0, 1, 0, 0x50000u);
  ExitProcess(0);
}
```

図３　破壊活動に関連した処理を行うコード領域


```
  v12 = 0;
  if ( ServicesReturned > 0 )
  {
    v13 = (LPCWSTR *)&lpMem->lpServiceName;
    do
    {
      v3 = OpenServiceW(hSCManager, *v13, 0x10000000u);
      hService = v3;
      if ( v3 )
      {
        dwBytes = 0;
        QueryServiceConfigW(v3, 0, 0, &dwBytes);
        v4 = dwBytes;
        v5 = GetProcessHeap();
        lpServiceConfig = (LPQUERY_SERVICE_CONFIGW)HeapAlloc(v5, 8u, v4);
        ChangeServiceConfigW(hService, 0xFFFFFFFF, 4u, 0xFFFFFFFF, 0, 0, 0, 0, 0, 0);
        if ( QueryServiceConfigW(hService, lpServiceConfig, dwBytes, &dwBytes) )
          PathRemoveArgsW(lpServiceConfig->lpBinaryPathName);
        v6 = lpServiceConfig;
        v7 = GetProcessHeap();
        HeapFree(v7, 8u, v6);
        GetLastError();
        CloseServiceHandle(hService);
      }
      ++v12;
```

図４　全てのサービスを停止させる挙動

以下は、ファイル共有されているファイルを上書きする処理ですが、先頭から一部だけ上書きすることでファイルヘッダを破壊しつつ書き込みサイズを抑え効率的にファイルを破損させていきます。ファイルを破壊する際、ファイルの内容を事前に読み込む処理がないため、復元を予定した暗号化等の処理ではなく単純な破壊目的であることがわかります。

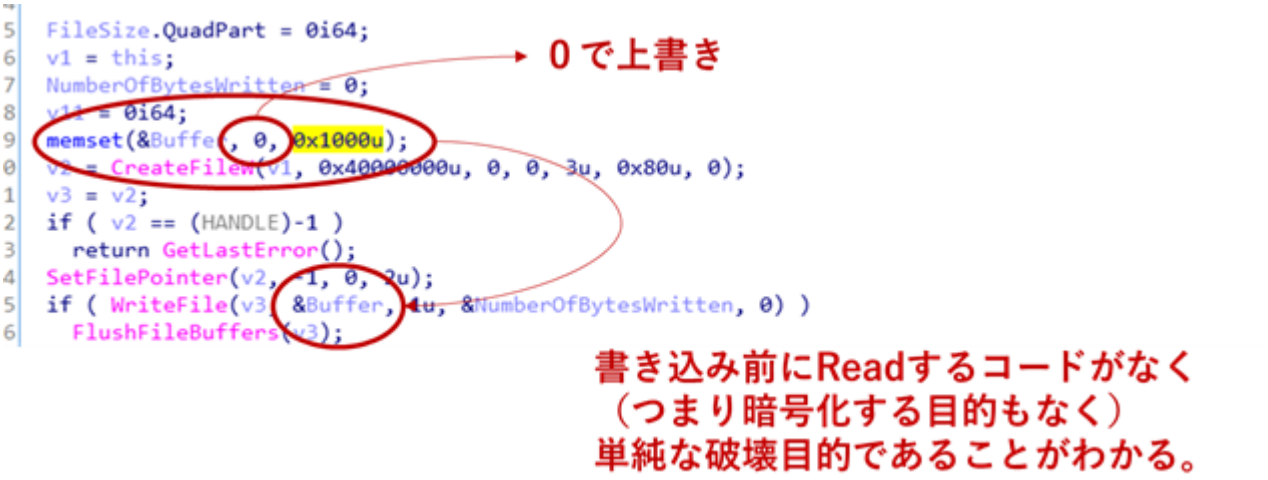


図５　ファイルを一部のサイズだけ０で上書きする処理

一方Olympic Destroyer本体は、上記（４）のEXEファイルの動作と並行して自身の隠蔽（消滅）を行います。具体的には、正規プログラムである「notepad.exe」を隠し状態で起動しWriteProcessMemoryによりコードインジェクションを実施、CreateRemoteThreadで「notepad.exe」内に書き込まれたコードを実行します。そのコードにより、Olympic Destroyerの本体が削除されます。（なおその際、Olympic Destroyerの本体は意味のないデータで内容を上書きされた上で削除されます。これには、フォレンジックを困難にさせる目的があると推測されます。）その後、notepad.exeは終了します。

図6　notepad.exeにインジェクションされたコードが本体を削除する部分

Olympic Destroyerの主な挙動は以上であり比較的シンプルなものとなっています。

摂取した情報をC&Cサーバ等へ流出させる動作やバックドアに類似するような挙動などは持ち合わせていません。また、明確な破壊活動と容易に判断できるようなMBRの破壊や大量ファイルの暗号化等の挙動なども持ちません。これらのことから、あくまで感染したコンピューターに「システム障害」が発生したと思わせ被害者側を混乱させる目的等があったと推測することができます。

なお一連の感染が行われた後、コンピューターは起動させると以下のようにOSの起動画面とブルースクリーンが繰り返し表示され正常に使用することが困難となることを確認しています。

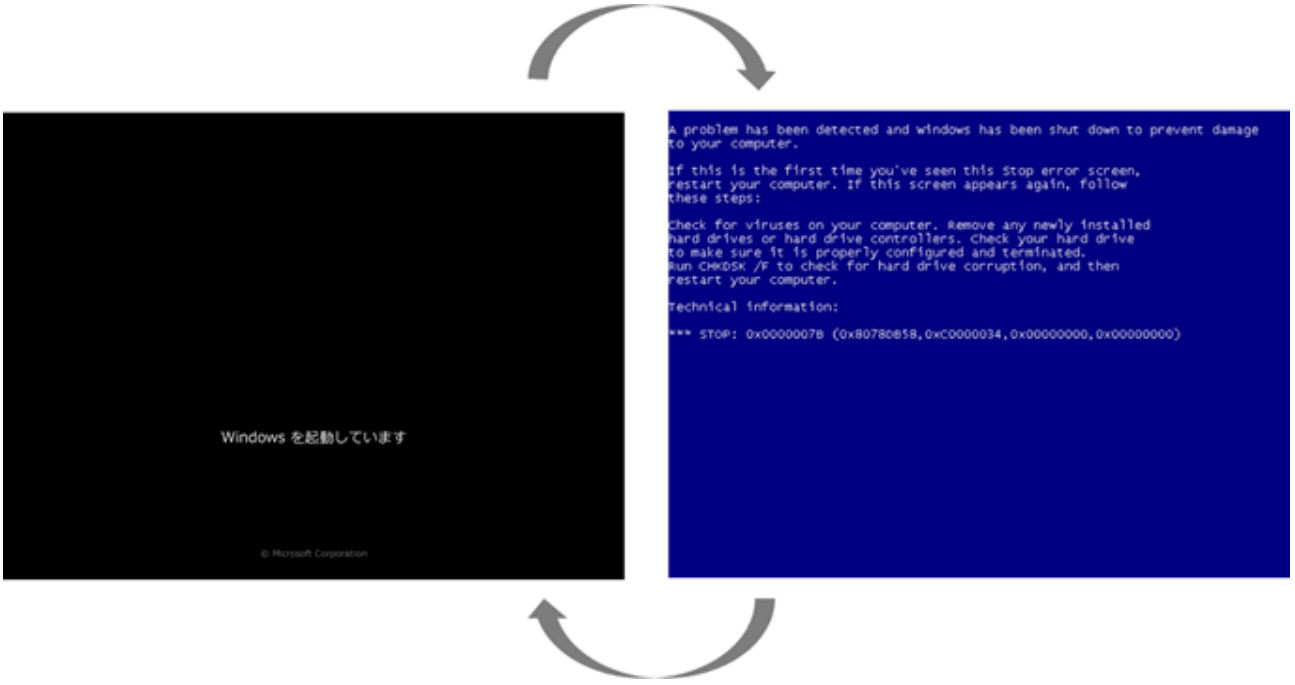


図７　Olympic Destroyerに感染しシャットダウンした後のPCの起動様子（Windows7 x86環境）

解析で見えた注目すべきポイント１（管理者権限で実行される前提の作りと侵入経路）

Olympic Destroyerは、LSASSからの認証情報窃取やvssadmin等を利用した破壊活動の挙動がありますが、これらの操作の一部は管理者権限でないと実行することができません。また一方で、実行された時点で自身のプロセスが管理者権限を持っていない場合UACを表示させるなどの権限昇格を要求する挙動も見られません。つまり一般に確認されているOlympic Destroyerの検体ははじめから管理者権限で実行されることを前提とした作りになっているようにみえます。本記事執筆時点でOlympic Destroyerの侵入経路は明らかになっていませんが、管理者権限での動作が前提となっていることを考慮すると、Olympic Destroyerには管理者権限を持ったさらに一段階上の階層となるドロッパー（親検体）がいたか、脆弱性を悪用した権限昇格による攻撃を用いた侵入経路が想定できます。ただし、管理者権限の実行権限がなくとも（権限がないことにより一部の処理に失敗しつつ）表面上はエラーなく一部の動作（保有しているパスワードを用いた横展開等）は継続できることから、完全に動作しないというわけではない点に注意が必要です。

解析で見えた注目すべきポイント２（取得したログイン情報の自身への埋め込みによる利用）

現在一般に入手できるOlympic Destroyerの検体には、複数のログイン情報と思われる文字列がバイナリに直接埋め込まれており、その多くが「Pyeongchang2018」というドメイン名と思われる文字列を含んでいることがわかっています。

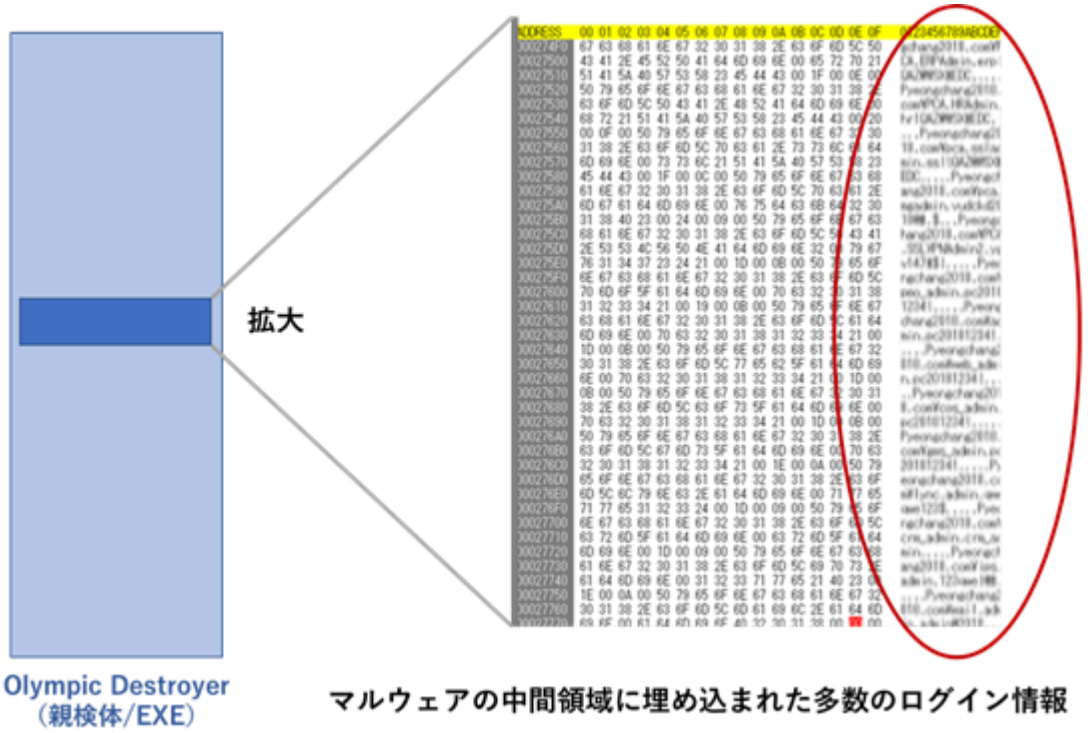


図８　埋め込まれたログイン情報の概念図

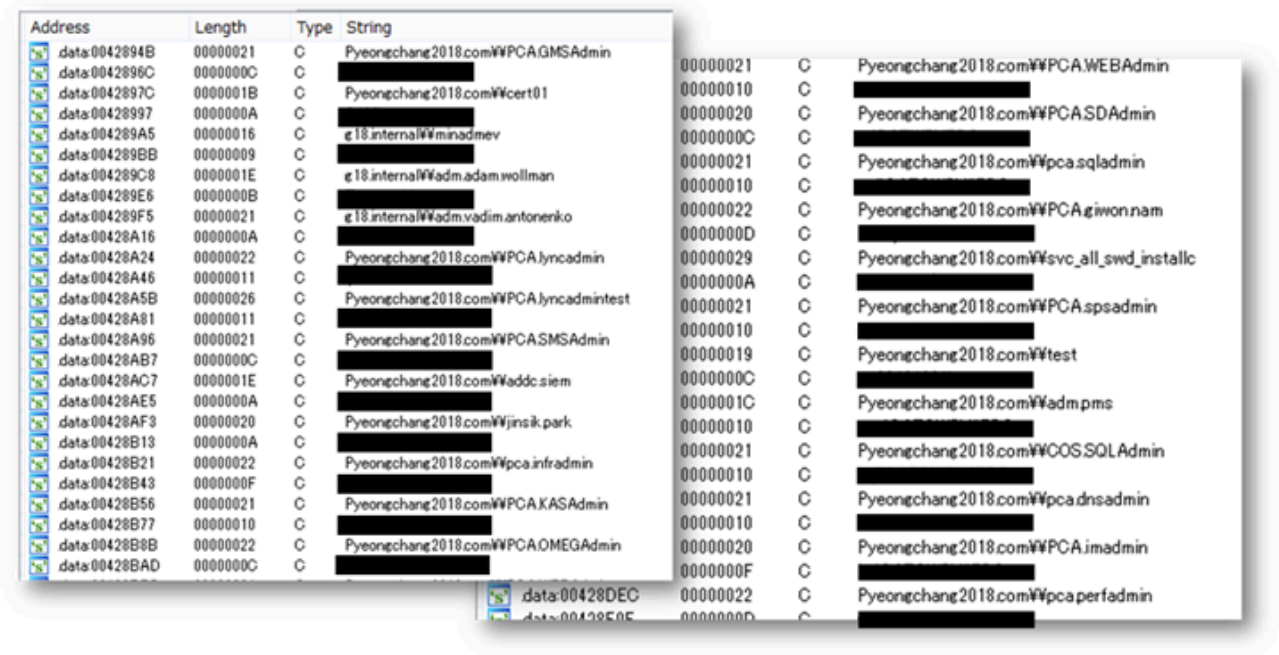


図９　「Pyeongchang2018」という文字列が多くみうけられるログイン情報

我々の調査の結果、Olympic Destroyerは感染時に窃取したログイン情報（ユーザ名とパスワード）の文字列を自身のバイナリの中間領域に埋め込んでいく挙動があることを確認しました。

窃取情報の埋め込みがバイナリの末尾ではなく中間領域であることからファイルサイズは変化しないため、見逃してしまう可能性がある挙動の一つです。

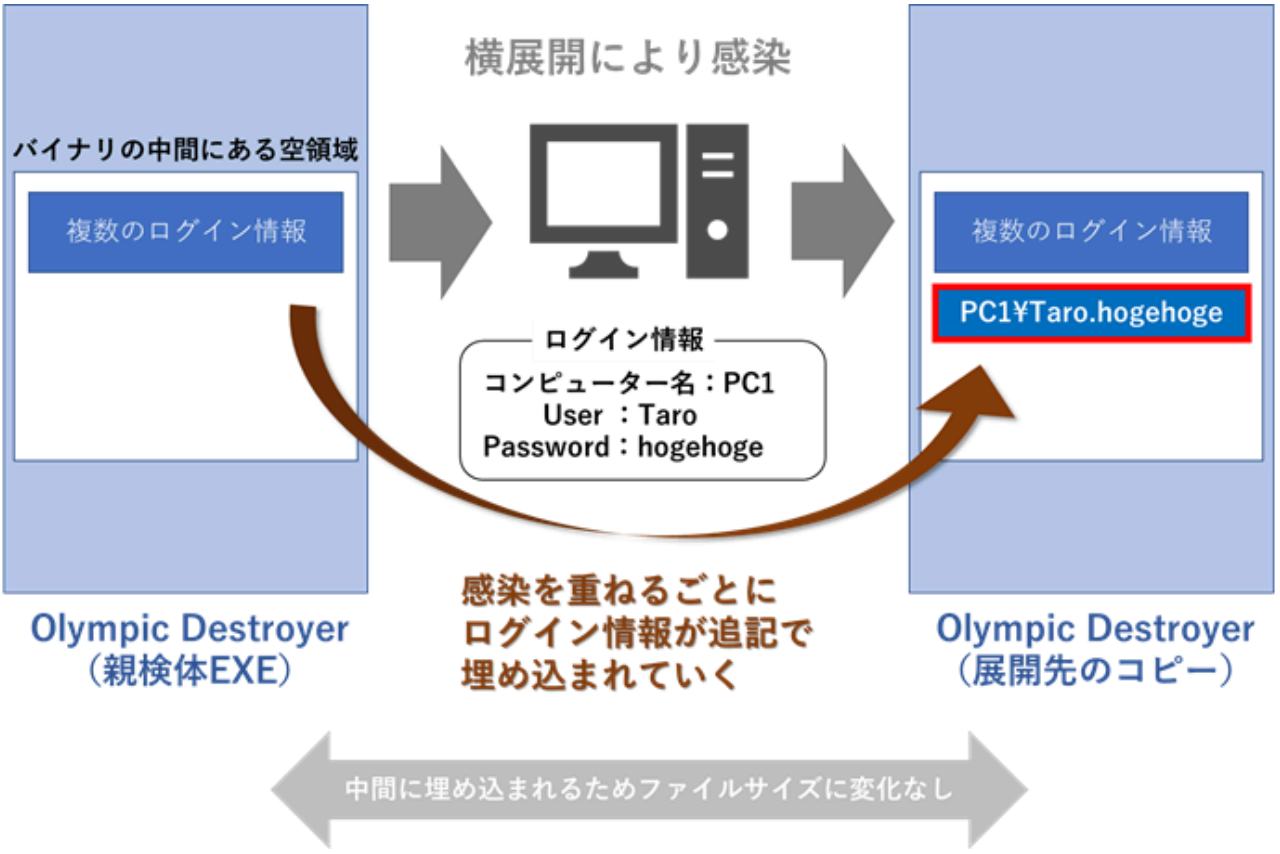


図１０　Olympic Destroyerが感染するたびにログイン情報を埋め込んでいく概念図

以下は、実際に我々の検証環境で感染させた際の実行前後のOlympic Destroyerの差分結果ですが、パスワードダンプツールにより摂取した情報を自身の内部に追記埋め込みしていることがわかります。

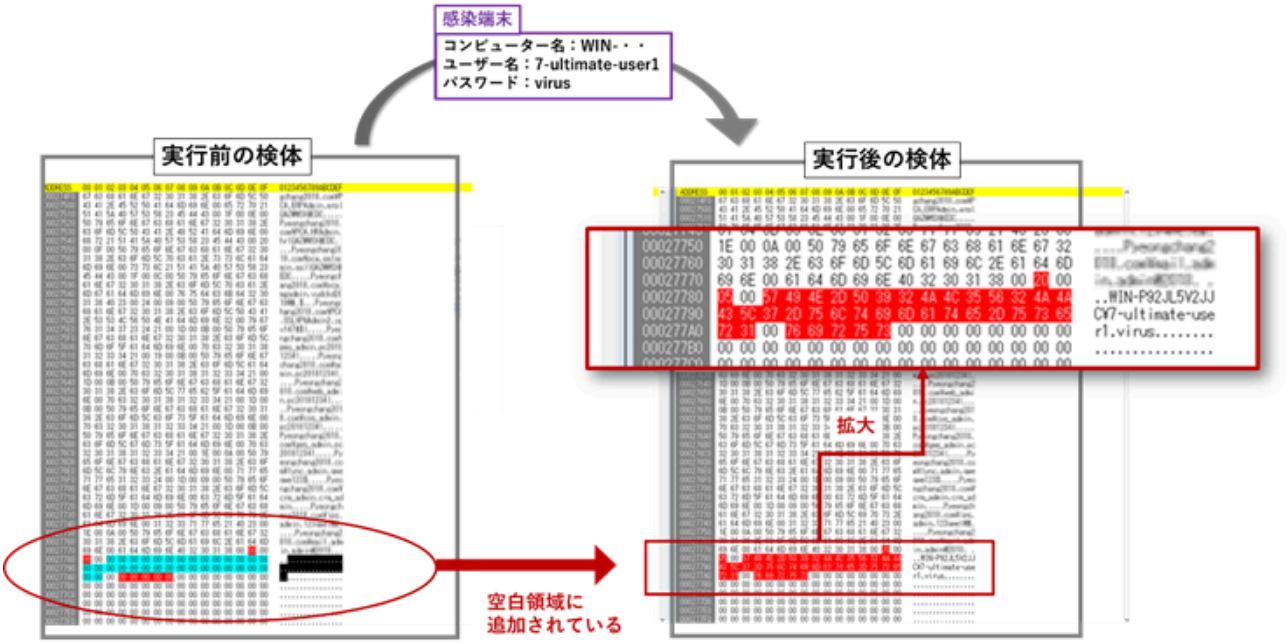


図１１　Olympic Destroyerが感染するたびにログイン情報を埋め込んでいく実証図

つまり、感染が広まっていくに従いマルウェアのバイナリに埋め込まれたログイン情報が増えていくことを意味します。この動作から、当初から一見ハードコーディングされたようにみえた複数のログ

イン情報は全て感染を重ねることで窃取され蓄積されたものと推測することができます。この推測が正しいとした場合、Olympic Destroyerのバイナリに埋め込まれたログイン情報の数を確認することで、おおよそどの時点の拡散検体なのか、また、初期感染時の端末を追う手掛かりになるかもしれません。

その他の情報

昨年度はWannaCry、NotPetya、BadRabbitなど世界的に混乱を招く大きなマルウェアインシデントが発生しましたが、それらの検体を解析してきた観点でOlympic Destroyerを見てみると、BadRabbitおよびNotPetyaに類似する挙動が一部垣間みえます。

例えば、以下は名前付きパイプによるプロセス間通信を用いた子プロセスとの連携に関わるコード部分の比較ですが、リソース領域から一時的に必要なEXEを%temp%に復号し、GUIDをパイプ名に利用した名前付きパイプによるプロセス間通信により必要な情報のみを取得して一時利用するといった処理の流れがほぼ同一です。その他にも、リソースセクションを用いた必要ファイルの格納および展開利用のテクニックや、PsExecを用いた横展開などいくらか気になる類似点があります。

図１２　NotPetya、BadRabbit、Olympic Destroyerのコード比較

当然、挙動やコードを類似させようとして意図的に作り上げることは可能なため、これらをもって何かを断言できることではありませんが、興味深い調査結果の一つであるといえるでしょう。

※本記事で使用した検体のハッシュ値は以下となります。

edb1ff2521fb4bf748111f92786d260d40407a2e8463dcd24bb09f908ee13eb9

吉川 孝志 の他のブログ記事を読む

Source: https://www.mbsd.jp/blog/20180215.html